#88 February 1984 $2.95 ($3.50 in Canada)

Dr. Dobb's Journal

For Users of Small Computer Systems

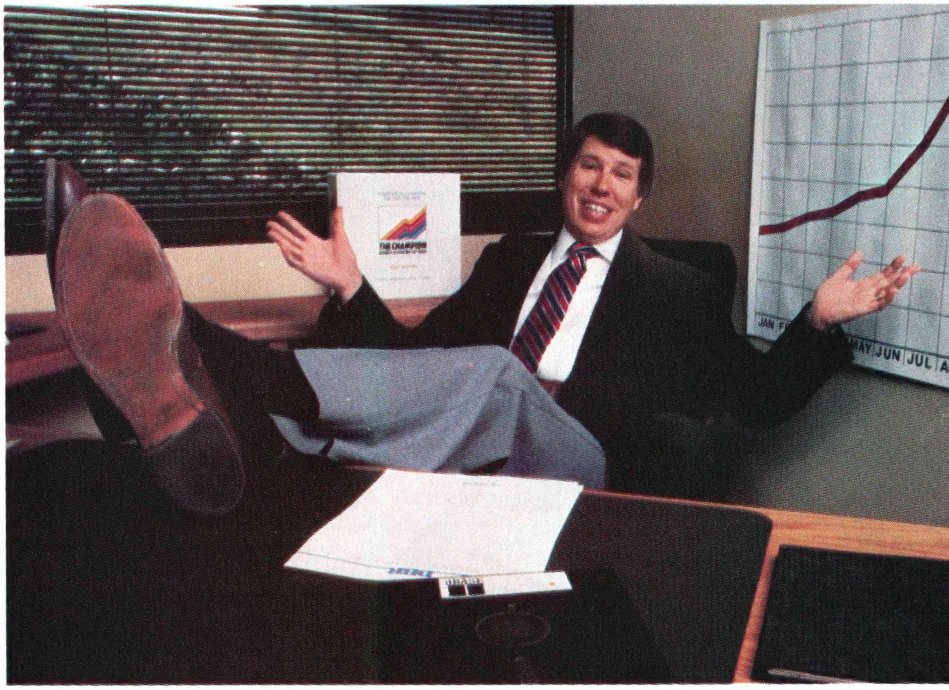
Virtual Personal Computer

Micro to Mainframe Connection

Communications Protocols

"dBASE II® is far, far better than a shoehorn."

Rusty Fraser
President
Data Base Research Corp.



"We laughed when our customers asked us to put our minicomputer-based real-time accounting system, The Champion,™ on a micro.

"No way was it going to fit, we thought.

"We'd have to create our own database management system and, even then, it'd be a tight squeeze.

"Then we discovered dBASE II, the relational database management system for microcomputers from Ashton-Tate."

"dBASE II was a perfect fit."

"dBASE II is a program developer's dream come true. The dBASE II RunTime™ module quickly provided us with the powerful text editing, data entry speed and other 'building block' capabilities we needed to develop and deliver a new Champion to our customers—the leading real-time on-line accounting system available for a micro."

The short cut to success.

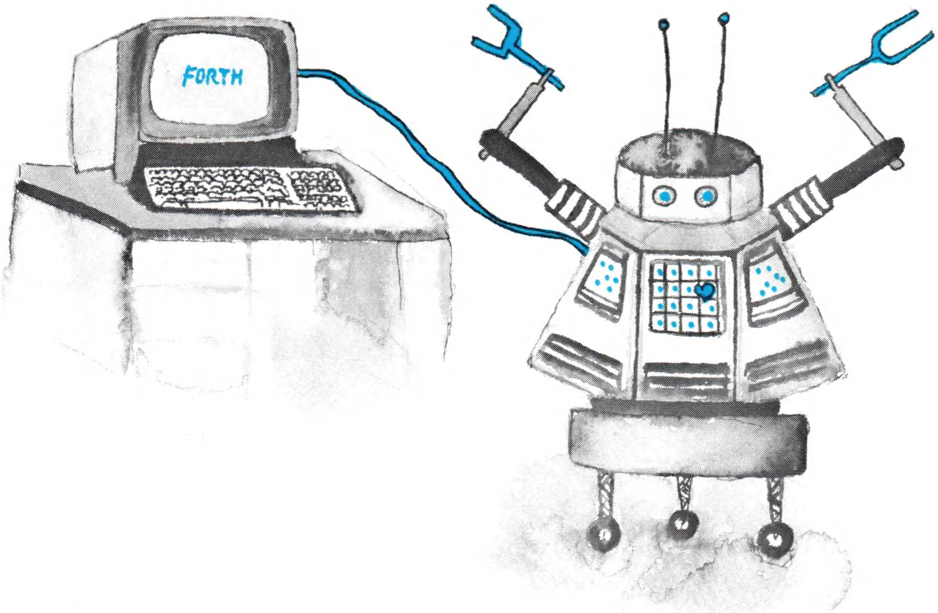
The dBASE II RunTime module has helped a lot of program developers like Data Base Research become successful software publishers.

For more about dBASE II and RunTime, contact Ashton-Tate
10150 West Jefferson Boulevard,
Culver City, CA 90230, (800) 437-4329,
ext. 217. In the U.K., call (0908) 568866.

For more about The Champion,
call Data Base Research at (303) 987-2588.

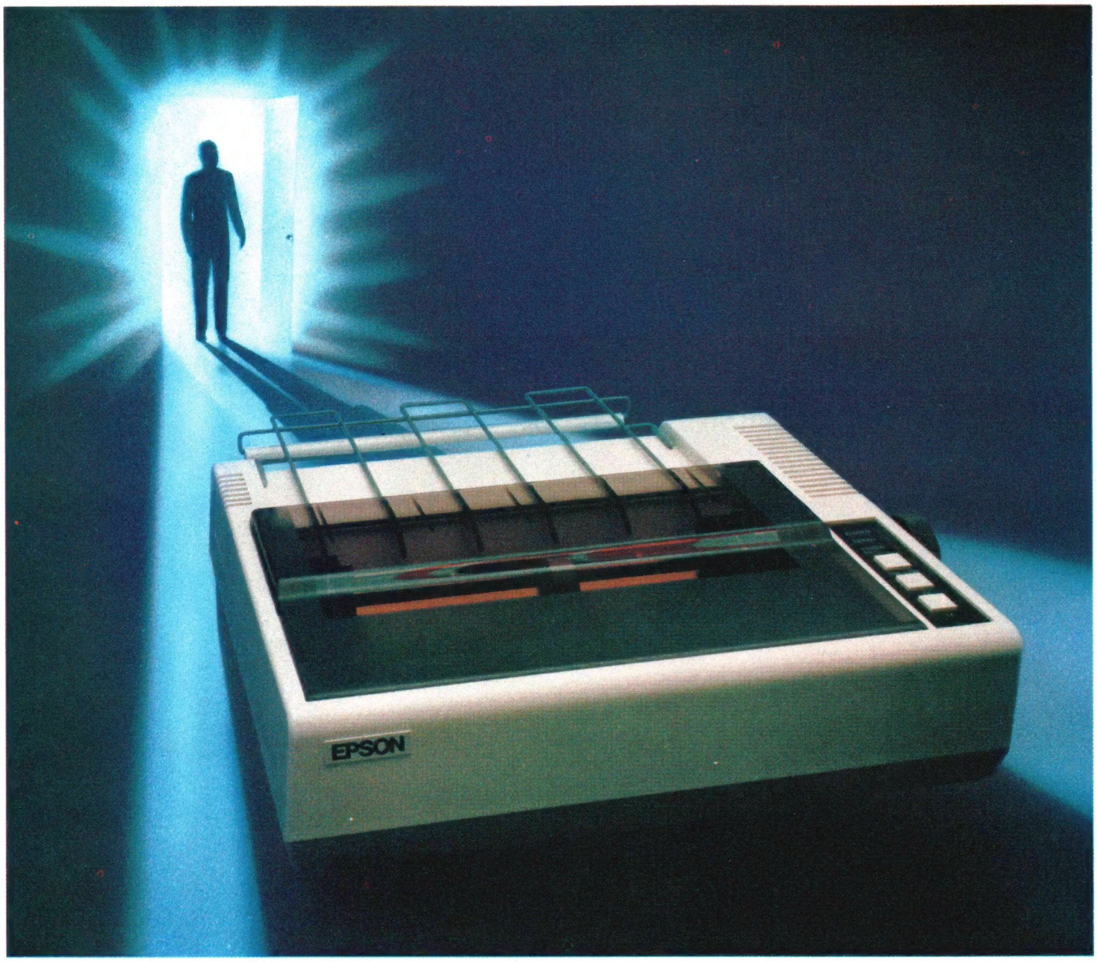
ASHTON·TATE

Circle **no. 3** on reader service card.



Circle no. **38** on reader service card.

Introducing the new RX-80.



A printer this good could only come from one company.

Epson.

Take a look at the new Epson RX-80 with a critical eye and you'll know what we mean. Epson printers are put together with manufacturing tolerances in the micron range. And meticulously checked for quality — one at a time.

Our reliability rate is approaching perfection. Which means you'll probably never — ever — have a problem with your RX-80.

Frills at a no-frills price.

Just because the RX-80 is affordable doesn't mean we've left off the goodies. You get 128 different typestyles with a resolution so sharp you have to look twice to make sure it's dot matrix. You get bidirectional printing, logic seeking, and a speed of 100 characters per second. And you even get programmable forms control, graphics, and a special Quiet Mode to keep the noise down.

This is your printer.

No matter what computer you own — or will own — do yourself a favor and make sure your printer is an Epson RX-80.

We build them to be the finest printers on earth. No matter how small the price.

Number one.
And built like it. EPSON
EPSON AMERICA, INC.
3415 Kashiwa Street, Torrance, CA 90505

Please call (800) 421-5426 for the dealer nearest you. In California (213) 539-9140

Circle **no. 27** on reader service card.

Dr. Dobb's Journal

For Users of Small Computer Systems

February 1984
Volume 9, Issue 2

CONTENTS

ARTICLES

DEPARTMENTS

Publisher – Jane Nissen Laidley
Editorial
Editor – Reynold Wiggins
Managing Editor – Randy Sutherland
Contributing Editors –
Robert Blum, Dave Cortesi,
Ray Duncan, Anthony Skjellum,
Michael Wiesenberg
Marketing
Marketing Director – Beatrice Blatteis
Marketing Assistant – Sally Brenton
Marketing Coordinator – Ed Gueble
Advertising
Advertising Director – Beatrice Blatteis
Advertising Sales –
Alice Hinton, Walter Andrzejewski
Circulation
Circulation Director – Terri Pond
Circulation Assistant – Billie Greenwood
Production
Art Director/Production Manager –
Barbara Ruzgerian
Production Coordinator –
Shelley Rae Doeden
Production Assistant – Alida Hinton
Typesetter – Paula Fairchild
Cover Design – Barbara Ruzgerian
on Via Video System One
Special Thanks – Berkeley School of
Computer Graphics, for technical
assistance

Subscription Rates: $25 per year within the United States; $44 for first class to Canada and Mexico; $62 for airmail to other countries. Payment must be in U.S. Dollars, drawn on a U.S. Bank.

Donating Subscribers **Contributing Subscriber:** $50/year ($25 tax deductible). **Retaining Subscriber:** $75/year ($50 tax deductible). **Sustaining Subscriber:** $100/year ($75 tax deductible). **Lifetime Subscriber:** $1000 ($800 tax deductible). **Corporate Subscriber:** $500/year ($400 tax deductible, receives *five* one-year subscriptions).

Contributing Subscribers: Christine Bell, W. D. Rausch, DeWitt S. Brown, Burks A. Smith, Robert C. Luckey, Transdata Corp., Mark Ketter, Friden Mailing Equipment, Frank Lawyer, Rodney Black, Kenneth Drexler, Real Paquin, Ed Malin, John Saylor Jr., Ted A. Reuss III, InfoWorld, Stan Veit, Western Material Control, S. P. Kennedy, John Hatch, Richard Jorgensen, John Boak.
Lifetime Subscriber: Michael S. Zick.

Foreign Distributors **UK & Europe:** Homecomputer Vertriebs HMBH 282, Flugelstr. 47, 4000 Dusseldorf 1, West Germany; Euro Computer Shop, 182 Rue du Faubourg St. Denis, 75010, Paris, France; La Nacelle Bookstore, Procedure D'Abonnement 1-74, 2, Rue Campagne – Premiere, F-75014, Paris, France; Computercollectief, Amstel 312A, 1017 AP Amsterdam, Netherlands. **Asia & Australia:** ASCII Publishing, Inc., 4F Segawa Bldg. 5-2-2, Jingumae, Shibuya-Ku, Tokyo 150, Japan; Computer Services, P.O. Box 13, Clayfield QLD 4011, Australia; Computer Store, P.O. Box 31-261, 22B Milford Rd., Milford, Auckland 9, New Zealand. *(Write for Canadian Distributors)*

EDITORIAL

This month we are pleased to present a special issue on telecommunications. We appreciate the response to the issue and regret that we could not include all of the material that was submitted. We will try to present other items on the subject in later issues. Here, however, we hope that you will find an interesting variety of topics.

As was noted briefly in the VPC article, the complete code listing was sufficiently large that we included only some of the illustrative routines. We are willing to publish the rest of the code, however, if there is sufficient interest. Those interested in seeing the remainder of the VPC level one system may drop the editors a note indicating this, or simply use the editorial response card included with the magazine.

*   *   *

Our referee program is now in place and working. As we explained in the initial call for volunteers, these referees are people who offer their thoughts and suggestions on material we are interested in publishing in *DDJ*. We are very please with the useful comments we are receiving, and we extend our thanks to all those who have offered their services.

This is the first issue in which we have really made use of the new referees. While we will publish a complete list of the board of referees next month, we would like to thank the following members for their timely insights on this issue:

Robert Blum,
Contributing Editor, DDJ
Patrick Burnstad, Lands End Computers
Keith Coye, Software Consulting Corp.
Mel Cruts, Triology
A. Gomez, Telecomp, Inc.
David Kirkland, Baker & Botts
Patrick Lynch,
Tandem Computers Inc.
Darryl E. Rubin, Rolm Corp.
Joseph Sharp, Micro Science Assoc.
Charles Wilde, IEEE, ACM

My thanks also to David Harris of PCC's PCNET Project for his assistance.

*   *   *

We have a number of useful topics planned for the near future. Many people have called looking for a runtime library for Small-C v2, presented in the December 1982 and January 1983 issues of *DDJ*. This spring we will publish the long-awaited library. In addition, we hope to have some other items available for use with the compiler. You will of course see more of our usual fare – languages, tools, algorithms, and so on – as well as a few surprises. Next month, we begin a two part article on a public-key data encryption system.

Talk to you next month.

Reynold Wiggins

Writer's Guidelines: All items should be typed, double-spaced on white paper. Listings should be produced by the computer, using a *fresh, dark ribbon* on continuous white paper. Please avoid printing on perforations. Requests to review galleys must accompany the manuscript when it is first submitted. Authors may receive a copy of the complete writer's guidelines and the current compensation schedule by sending a self-addressed, stamped envelope.

Dr. Dobb's Journal (USPS 307690) is published twelve times per year by People's Computer Company, P.O. Box E, Menlo Park, CA 94026. Second class postage paid at Menlo Park, California 94026 and additional entry points. Address correction requested. Postmaster: send form 3579 to People's Computer Company, Box E, Menlo Park, CA 94026 · 415/323-3111
ISSN 0278-6508

LETTERS

More Forth Writeability

Dear Dr. Dobb,

Before I begin, this is a *letter,* not an article – so please don't pay me yet! I received my first *DDJ* yesterday (whoopee, my own copy) and was not disappointed.

I was delighted that you decided to reprint Harvey Glass's article "Towards a More Writeable Forth Syntax" and thus make it available to us masses. There is a growing awareness, despite the jealous attempts at standardization by the purist elite, that Forth is simply not readable (or writeable) enough for broad acceptance. A group of languages is being developed in an attempt to make it more palatable and consistent. These include STOIC (J. M. Sachs, S. K. Burns, "STOIC – An Interactive Programming System for Dedicated Computing," *Software – Practice and Experience*, Vol. 13, 1-16, 1983), PISTOL (E. E. Bergmann, "PISTOL – A Forth-like Portably Implemented STack Oriented Language," *DDJ*, No. 76, Feb. 1983), and a modification of PISTOL which I am in the process of developing called REPTIL.

The most important departure of these languages from Forth is that a string is also a fundamental data type. Thus every name begins life as a lowly string and can be subsequently manipulated or converted to a more elite *verb* which "does" something by simply assigning it an action using the colon defining verb. In REPTIL, for example, a definition for SQUARED could be:

```
    'SQUARED DOES: DUP * :END
```

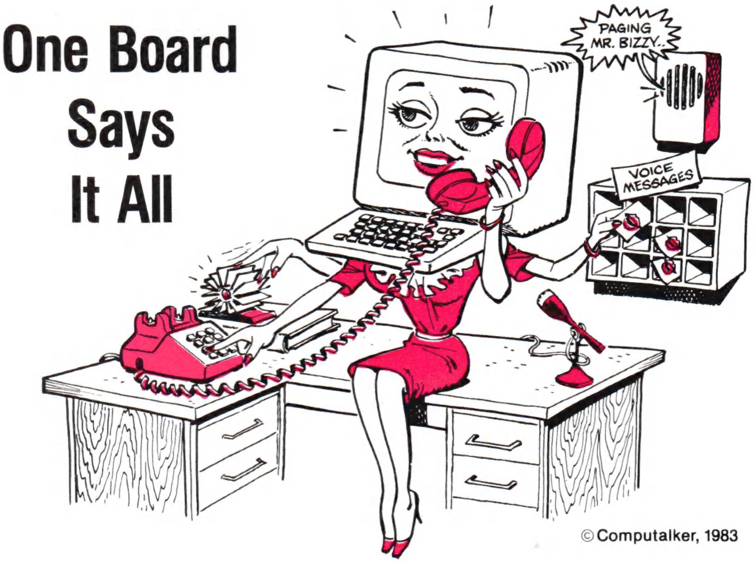
This is a true "Reverse Polish" approach. Forth defining words have an inconsistent "infix" format, thus:

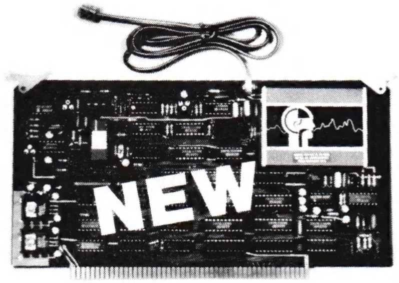
```
            : SQUARED DUP * ;
```

The second major departure is that compilation is done on a line basis rather than compile and execute on a word-by-word basis. In some cases execution can be deferred over a few lines. The various structured constructs are thus allowed in both immediate and colon definition modes.

The syntax proposed by Harvey Glass is delightful – it considerably improves readability. I don't believe that Reverse Polish is less natural than Infix; in fact, in a recent article on Creole languages (D. Bickerton, "Creole Languages," *Scientific American*, July 1983) we see that postfix notation, "The poor people all potato eat," or prefix notation, "Work hard these people," is more "natural" than our formalized Infix heritage. What, then, is the source of Forth's lack of readability?

One reason is the cryptic symbolism, where nondescript characters like "@" "!" "," and "." take the place of more suggestive verbs *fetch, store, include,* and "=" (or *pop-and-display*). Another reason is that most humanoids prefer to associate their objects with names, rather than have them nebulously floating around on stacks. In Forth-type languages, however, all constants and variables are global, and their use thus considerably clutters up the structured modular nature of the programs. The concept of a local parameter, as well as the use of NOP's in the form of parentheses and commas, is a





major breakthrough in this matter.

The use of parentheses has been recently proposed by Bergmann (E. E. Bergmann, "Languages & Parentheses" – *planned for a future issue of DDJ – Ed.*) for use in Forth-like languages, however not as simple NOP's. They are controlled by syntax checking in that execution is deferred until matching parentheses are established. The comma will surely upset the purists, as will the use of <DEFINE in place of ":". How can we throw away the cryptic ethnic symbolism of Forth with such disregard of tradition? Easily!

The REPTIL equivalent representation of the Ackermann function presented by Harvey Glass would be as shown in Figure 1 (at right). A few words of explanation are in order. All relational operators ask the question: Is it true or false? Thus they are all followed by a question mark, e.g., =0? – is TOS equal to zero? The wording of the conditional branch is designed for readability; I cannot accept the topsy-turvy Forth "IF – ELSE – THEN" hence they have been respectively replaced by "?THEN – ?ELSE – ?END."

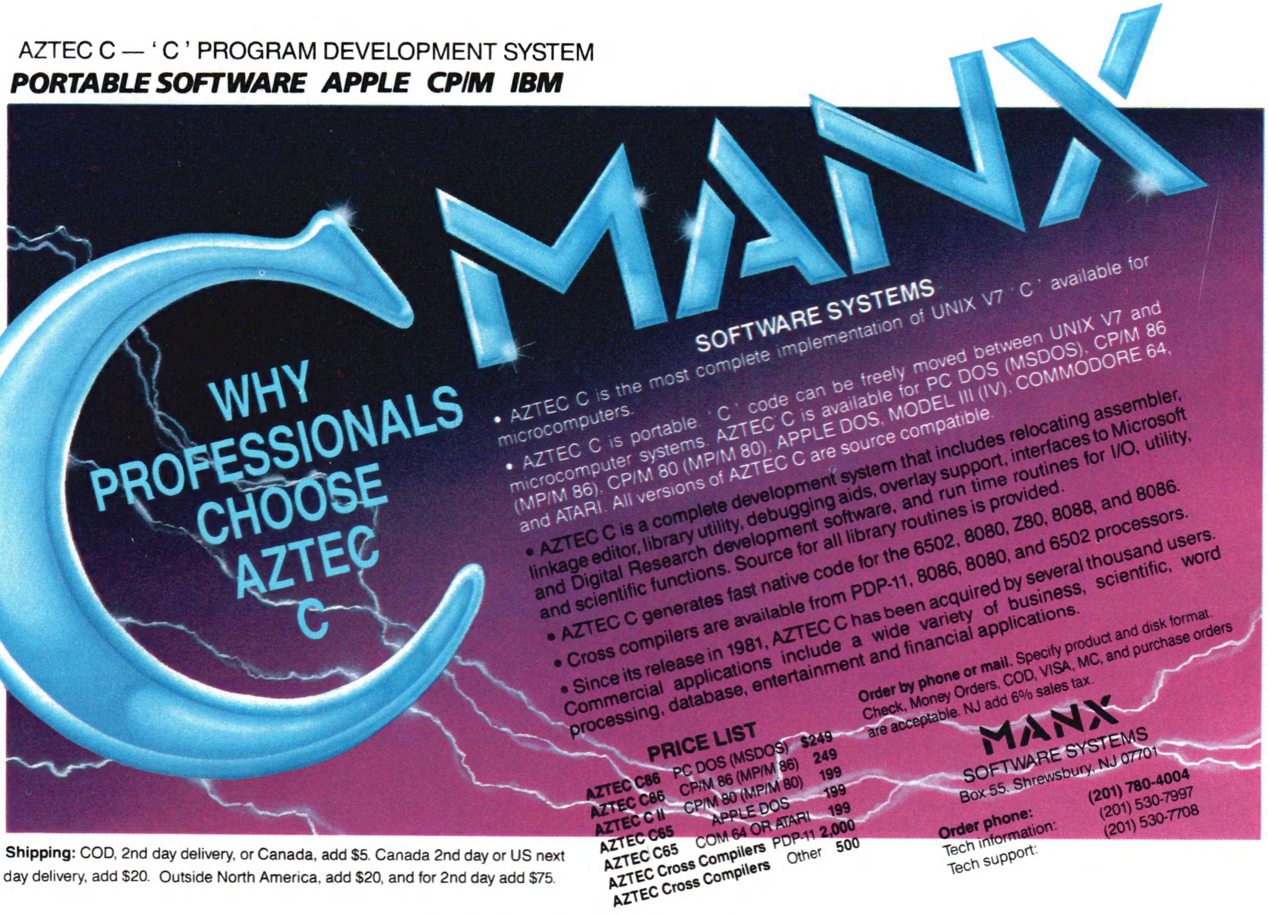
The parentheses are those proposed by Bergmann and the comma is the NOP of Glass. STOIC originally proposed the vocabulary stack technique, in which, for instance, ASSEMBLER< will push the assembler vocabulary on the vocabulary stack and > pops it. Thus LOCAL> is a system vocabulary which does not have a permanent branch point. When invoked, it is linked to the latest *current* vocabulary.

:IS defines a constant (it *is* a value), and :HAS similarly defines a variable in the normal sense (it *has* a value, which can change); thus invoking I and J (defined as constants in the LOCAL< vocabulary) simply pushes their respective values to the stack.

Sincerely,
Issy Urieli
225 Highland Ave.
Athens, OH 45701

```
'ACK DOES:
    LOCAL<  'I  :IS    'J  :IS


  I  =0?   ?THEN                 { J  1  + }
           ?ELSE   J =0?  ?THEN
                                 { I  1  -  ,  1  }  ACK
                          ?ELSE
                                 { { J  1  -  ,  { I  ,  J  1  -  }  ACK  }  ACK  }
                          ?END
           ?END    >
 :END
```

Figure 1.
REPTIL Equivalent Representation of the Ackermann Function



Curious about Ackermann

Dear Editor,

The function defined by Harvey Glass in the Nov. '83 issue is not the same "Ackermann function" as defined by Petersen in *Forth Dimensions,* III, 3, pp. 89-90. (Peterson's version is shown in Figure 2 at the right).

The fig-Forth code which Petersen gives to implement this function is garbled. Glass's standard Forth code is better, but still not correct. A correct implementation of the above function can be had by replacing the DUP on the second line of Glass's Figure 1, with an OVER.

I am curious about the origin of this function. Who is Ackermann, and where did he publish a definition of this function? Of what use is the function other than testing recursion in programming languages? Can any of your readers help with these questions?

Having asked the above, I imagine you may be curious to know how I can be so sure that Petersen's definition is "correct" and Glass's is "incorrect." The answer is that Glass's definition leads to infinite recursion with an unbounded increase in the depth of the recursion for all nontrivial values of the arguments while Petersen's leads to large, but finite recursion.

Sincerely,
Paul E. Condon
260 Devonshire Blvd.
San Carlos, CA 94070

$$
\begin{array}{lll}
\mathrm{ACK}(\,I,\ J\,) & = J+1 & \text{for } I=0 \\
 & = \mathrm{ACK}(\,I-1,\ 1\,) & \text{for } J=0,\ I>0 \\
 & = \mathrm{ACK}(\,I-1,\ \mathrm{ACK}(\,I,\ J-1\,)\,) & \\
 & & \text{for } J>0,\ I>0
\end{array}
$$

Figure 2.
Petersen's Version of the Ackermann Function

Merci, Morrow

Dear Editor,

I purchased a Morrow Micro Decision from Priority One Electronics when they held their grand opening in Irvine. It is the first factory-assembled computer I have owned. For $2100, I got the computer with two double-density 5¼-inch drives (192K each), 64K RAM, WordStar, Basic, Microsoft BASIC, Correct-it, Logicalc, and Personal Pearl, the MDT 20 terminal, and COEX 80 printer.

The system worked wonderfully well for about a month, but then began occasionally to give the dreaded BDOS error on B. As it seemed to be getting worse, I loaded the system in my car, and drove to Irvine. At the store, I set it up, and of course everything worked perfectly. For about three hours I thrashed it on every piece of software it had failed on, to no avail. So I bought some new disks and a disk head cleaning kit and loaded it all back in the car.

Once home everything worked well again for almost two months. The key phrase is "almost two months." To be exact: 89 days from the date of purchase drive B quit. As I did not wish to drive from San Diego to Irvine again, I called and asked about warranty repair. Priority One in Irvine told me to call Morrow. I really wasn't expecting a cheerful reception from Morrow, but the lady I spoke with was extremely courteous and asked if I had contacted the dealer. I replied "yes," and then she said she would call me right back. About 20 minutes later she did and asked if I could mail the computer in. Of course I could, but I explained that I am in the Navy, and asked if she could have it back in less than three weeks since I was going overseas. Well, I know that things take time. I really didn't expect to have it back, but I wanted to have the computer overseas with me. Can you believe they would try?

To shorten this up, they did. And even a note inside telling me my diagnosis of the problem was correct. I am amazed. Of course you probably have heard good things about Morrow before, but I can't even get that kind of service on my $16,000 car. I wanted to make sure they got their kudos. The computer has been with me now for four months on a ship overseas, and that means vibration and infrequent use. The little jewel has held up marvelously – in fact, so well that I would seriously investigate nearly anything before hardware for a fault. I haven't even named this one yet, maybe Clark, for Clark Kent – you know, Superman in plain clothes.

In Navy talk, "Bravo Zulu" for a super job, Morrow.

ETC Howard L. Howell
USS Fletcher DD992
FPO San Francisco, CA 96665

A Plea for Xitan Help

Gentlemen:

I don't know how to tell you how much help you have been to me in my constant learning about computers. Even when I don't understand 90% of the magazine, I read through. Each month it gets easier.

Right now I could use a favor from one of your readers. I am the proud owner of a Xitan computer. That's right, Xitan. The trouble is the documentation needed for repair must still be the property of the first owner. I sure don't have it.

If anyone out there has the schematics, etc., for a Xitan, please let's get together.

Thank you for being there.

Yours very truly,
Larry Litwin
1935 La Habra Blvd.
La Habra, CA 90631

HELP for Readers

Dear Editor:

I wish to announce that the Small-C Help Facility, as published in the October 1983 issue, is now available from Pyramid Systems, Inc. of San Marino, California. Those who wish to use the package but don't relish typing it in by hand ought to appreciate the low cost nature of this source.

Thanks to those who have written concerning the package. I will be happy to keep *DDJ*'s readers informed of any known bugs and fixes.

Sincerely,
John Staneff
Route 1, Box 801
Ellensburg, WA 98926 DDJ

DR. DOBB'S CLINIC

by D. E. Cortesi, Resident Intern

System Languages

We seem to have hit a community nerve with our blast at current compilers; some of you are actually writing. Great! Let's review somebody else's opinions for a change.

Most everybody agrees that we need a portable language for systems programming, in an implementation that produces efficient object code. Even Steve Newberry, who markets a structured assembler, likes the idea. He writes, "I suggest you take a look at Leor Zolman's BDS C (for the 8080) and Mark De Smet's C88 (for the 8086). They are so good that it is often hard to justify the use of structured assembly packages like my SAL/80. But there are, and I believe will long continue to be, situations in which . . . a high level language is simply not appropriate for the job. In those cases the structured assembly language provides the most economical solution."

Thanks for the tips, Steve. BDS C scores high in most published benchmarks and has an enthusiastic user group. However, it differs from the Unix standard at a number of points. The advantage of Aztec (and perhaps other C compilers) is its close match to Unix C. Standardization allows program portability, and that's supremely important in today's software market. Here's another point: most textbooks on C are based on the Unix standard. When a compiler deviates, it puts impediments in the way of the independent student. It's hard enough to learn a language on your own without having to translate the examples in the book.

Jim Howell of San Jose, CA, compiled our sort program under Introl C, using OS-9 on a 1 MHz SS-50 system. He reports a linked object file of about 10K bytes (versus our 16K) and execution times that were comparable (except for writing, which was only slightly slower than reading). He notes that our IGNORAD constant might well have been required because of a blank line at the end of the input file. Very possible.

David King of Woodside, CA, did an informal benchmark comparing Aztec C, Microsoft's BASCOM, and Digital Research's PL/I-80 and CBASIC. The test program leaned hard on floating point accuracy and looping. The fastest execution time came from PL/I-80 (10 seconds versus 15 for BASCOM, 16 for CBASIC, and 19 for Aztec C). PL/I and BASCOM produced 9K object files, versus Aztec's 11K. Compile times were comparable. Only PL/I and CBASIC were able to produce the correct answers to the limit of their precision.

King takes off from these results to reach some conclusions on the relative merits of the languages involved. This is something we all have to be careful about. It is impossible to benchmark a *language*; one always benchmarks an *implementation*. Every aspect of compiler performance – portability, execution time, object size, compile time, and numerical accuracy – is a product of the implementation. We can live with just about any Algol-descended language for systems programming, so long as its implementation adheres to a standard and produces efficient object programs.

David Clark of State College, PA, sent a bulky package of helps for Dwight Irving, who was struggling to implement the UCSD p-System. With them, Clark sent some helpful comments on Aztec C:

"The failure to link the 'write' function that you mentioned is not an error. The standard I/O library includes that function for system-level disk I/O. The library and the 'write' function are both described in K & R.

"The slowness associated with redirecting output to a disk file is a problem with the Aztec version of the library. Console I/O (stdin, stdout, stderr) is normally done one character at a time. When redirecting output to a disk, the I/O is still done a single character at a time. Since no buffering is done, a sector must be read or written for each character. [*Aha!*] Most CP/M implementations perform redirected I/O by saving the characters in a buffer that is some multiple of the sector size. When the buffer becomes full or the file is closed, the entire buffer is written at once.

"The size of the code file is due to the way the library is implemented. When the library is created, all of the functions present in one file will be compiled into a single module. At link time, the library is scanned for needed functions. When one is found, its entire module is linked. I have two other versions of C that construct the library by compiling just one function at a time. They can then link a single function. Of course, building a library can be pretty tedious when you have dozens of functions to change. [*That's why God made submit files, David.*]

"There is a public domain rework of the Aztec library that remedies these two defects. Redirected I/O has been cleaned up and is much faster. The library files have been broken into much smaller pieces, too. The library is mostly the work of Herb Shulz and can be obtained from a friend of a friend of a friend. . . ."

Jack Purdum of Indianapolis, IN, has another perspective on the size of a C run-time library:

"I'm not unbiased on C compilers (we market one that competes with Aztec). However, all compilers suffer similar limitations in regards to a number of your comments. First, printf(), scanf(), and fprintf() are *very* large functions because they are required to do so many things. Your program uses only a fraction of the power of these functions. This 'H-bomb-to-kill-an-ant' problem comes with using built-in functions from the library. If code size is important [*when the heck is it not?*] writing a new function that does only part of what the complete function does is one solution – your xprint() is an example. We (Ecosoft) have compiler switches that allow an integer printf() to be used, thus avoiding linking the floating point routines. Another switch avoids the file I/O options when doing just screen I/O. While this helps, it's not a total solution."

You make an important point, Jack. The scanf() and printf() functions are defined to convert between *any* data type and ASCII. Furthermore, the conversion string argument that guides them may be a variable; ergo, the compiler can't know at compile time which conversions are actually required. As a result, a single call of

printf("hello")

may cause the linking of the library routines for conversion of signed and unsigned integers, longs, floats, you name it. And they link other support routines, so any printf() call will cascade just about the whole run-time library into your program.

We can think of two solutions. First, instead of your compiler switches, why don't you put the printf() and scanf() functions in a separate library, and distribute two versions of that library? One version would support only strings and short integers; the other would be full-function. Name the one you want in the linker command line. Use macro facilities cleverly enough, and the two versions of the functions could be compiled from the same source file.

Second, why do the standard functions have to be written in C? Sure, Drs. K & R showed model support functions in *The C Programming Language*, but aren't they to be taken as just that, models? Use them to get your compiler off the ground; use them in Beta-test; use them as executable functional specifications. But when it comes to the final version of the most heavily used code in every user program, shouldn't you hand-code them in the target assembly language? Isn't it the whole point of a high level language that one person, the compiler designer, takes on the burden of using the machine effectively, thus lifting that burden from the shoulders of many people? A hand-tuned assembly version of printf() ought to run rings around a compiled version, and the payoff, multiplied over thousands of user programs, more than justifies the effort.

Keep Talking

This discussion is based on our assertions that a portable, efficient, systems programming language is desperately needed, and that no existing microcomputer compiler is adequate to meet the need. We'd make a third claim: compilers are as poor as they are only because we users have not demanded better ones.

If you disagree with any of these claims, feel free to write. Write too if you have counter-examples: compilers that do produce efficient code, cases where assembly language is needed regardless, or proof that certain language features can't be made efficient no matter how much demand there is. Also, does anybody know who did the first optimizing compiler? It was a Fortran for the IBM 360 done in the late 60's, but we've lost the reference. What other optimizers have you used, and how well did they work?

Move Over, Ray

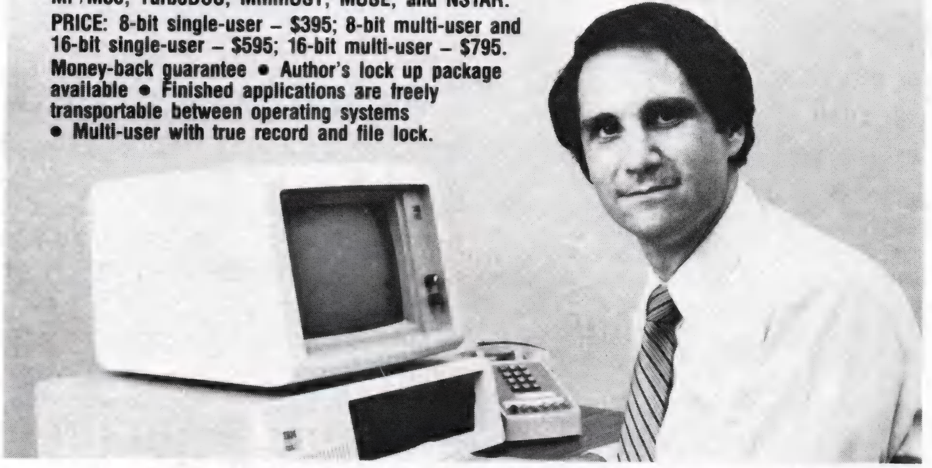
Last month we tromped all over Bob Blum's columnistic territory and promised, in a fit of naked ambition, to move in on Ray Duncan's this time (we columnists have strong territorial instincts). All this comes about because what goes into this column is – in the absence of reader input, hint hint – whatever happens to have passed over the columnist's desk or through his keyboard in the weeks preceding the deadline.

What was passing through our keyboard in recent weeks was the final draft of a book on Concurrent CP/M, in support of which Texas Instruments had very generously loaned a Professional Computer with CCP/M installed. The Professional Computer, or "Pegasus" as we insiders call it, worked very well, and its CCP/M did, too. This version of the system uses the "3.1 BDOS," the same level of file system as used in CP/M Plus. (You can tell the 3.1 BDOS by its use of the INITDIR command to set up disks for timestamping.) Its BIOS and system integration had been breathed on by DRI, which may explain why we found no bugs or performance problems in it.

There was a command missing, however. CCP/M does not implement the "file search path" notion of CP/M Plus. There is only a single "system drive" which will be searched when a command can't be found on the default drive.

There are various ways of choosing the system drive. The default choice is set when the system is generated. In at least the first edition of CCP/M for the IBM PC, the system drive is set automatically as the highest-lettered drive in the system. The generic version of CCP/M includes a command, SYSDISK, that will change the system drive. In the TI version of CCP/M, the system disk is either diskette drive A: or, if you have one, the hard disk on drive E:. The SYSDISK command was omitted; there is no way of changing the system disk.

There are times when you might want to change the system drive. If you have only diskette drives, the system



disk should at least be different from A:, the usual default drive. A: will be searched anyway, so the second search is wasted unless the system disk is another drive. Again, if you have an electronic pseudo-drive, you might want to make it the system drive after you have brought the system up and initialize it. We thought that the lack of a SYSDISK was an unnecessary handicap on the TI system.

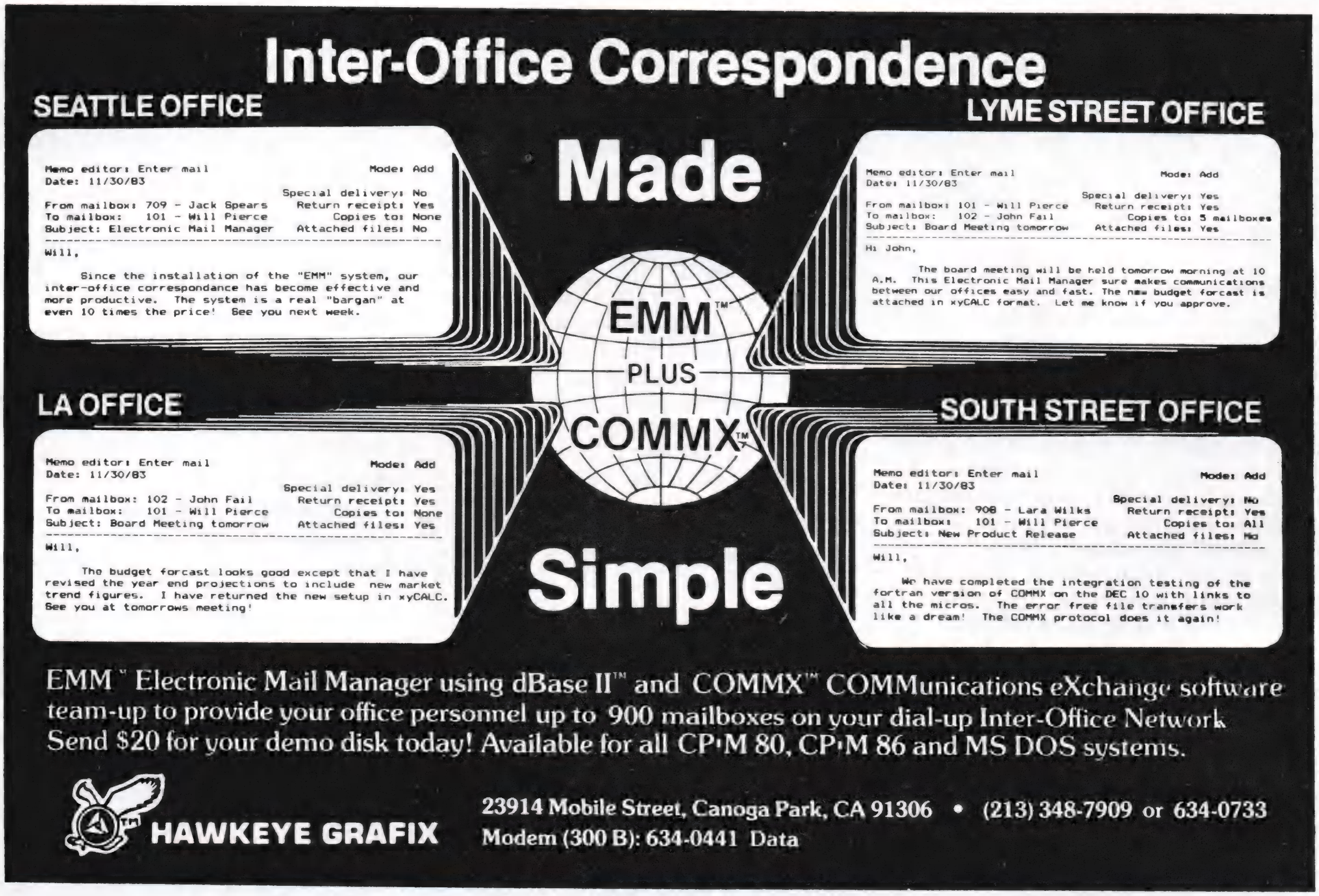
With a little research, we figured out how to change the system drive. A short assembly program does it by putting the drive number in the System Data Area (see Listing One, below). As a bonus, you can change four lines to create another program which will change the temporary drive. The temporary drive is the drive on which the system will build scratch files, especially the VOUT files of data from buffered consoles.

Reader Ballot
Vote for your favorite feature/article.
Circle Reader Service **No. 190.**

Clinic Listing (Text begins on page 10)

```
;================================================================
;
;        SYSDRIVE  [ d: ]
;
; SYSDRIVE is a simple, 8080-model, program that reports on and
; sets the system drive-letter under Concurrent CP/M.  Some
; CCP/M systems are delivered with a SYSDISK command that does
; the same job.
;
; Given no command argument, SYSDRIVE merely reports which drive
; is the system drive.  Given a drive-letter and colon, it will
; set that drive as the system drive.
;         THE DRIVE-LETTER IS NOT VALIDATED.
;
; The same source file becomes a TMPDRIVE command by changing
; the indicated ((operands)).
;
;================================================================
;
Bdos            equ     224 ; interrupt number for BDOS
C_WriteStr      equ     9   ; code to display a string
S_Sysdat        equ     154 ; code to get System Data Area
Sysdisk         equ     4Bh ; offset of Sysdisk in SDA
Tmpdisk         equ     50h ; offset of Tmpdisk in SDA
;
                CSEG    ; everything in the code segment
                ORG     005Ch
Operand_drive   rb      1 ; First-operand drivecode
                ORG     0100h ; code begins here
;
                mov     cl,S_Sysdat
                int     BDOS     ; BS:EX -> SDA
                mov     al,ES:Sysdisk[BX] ; ((ES:Tmpdisk[BX]))
;
                push    bx       ; save regs while..
                push    es       ; ..displaying msg
                add     al,'A'   ; make drive printable
                mov     msg1_drive,al ; and put in msg
                mov     cl,C_WriteStr
                mov     dx,offset msg1
                int     BDOS     ; display msg1
                pop     es       ; recover saved
                pop     bx       ; ..registers
;
                mov     al,Operand_drive
                or      al,al   ; was a drivecode given?
                jz      exit    ; (if not, stop)
;
                dec     al      ; 00=A, 01=B, etc
                mov     ES:Sysdisk[BX],al ; ((ES:Tmpdisk[BX]))
                add     al,'A'  ; make new drive printable
                mov     msg2_drive,al ; and put in msg
                mov     cl,C_WriteStr
                mov     dx,offset msg2
                int     BDOS    ; ..and print it
;
exit:           retf    ; far-return ends program
;
msg1            db      'System' ; (('Temporary'))
                db      ' drive is '
msg1_drive      rb      1       ; filled in from SDA
                db      ':',13,10,'$'
;
msg2            db      'System' ; (('Temporary'))
                db      ' drive set to '
msg2_drive      rb      1       ; filled in from operand
                db      ':',13,10,'$'
;
                end
```

End Listing



CP/M EXCHANGE

by Robert Blum

Over the last several months I have spent a little time talking about CP/M Plus and its new features. All of these discussions have been from the outside looking in, probing no deeper than just below the surface. Even from that point of view I was very excited about what CP/M Plus was built to do and its many new performance options.

CP/M Plus is now in daily use on my machine and is largely responsible for turning the tide against V 2.2, which has been retired to second string and bench warming. As is the case with most new products, there are a few rough edges that need to be smoothed out, but nothing more serious than cosmetic defe‹ts. I still have a lengthy wish list of features that I would like to have. However, my list now contains desirable additions, not items that are necessary to plug holes in an inadequate system.

I will be talking more about CP/M Plus regularly in future columns. Some of the areas that I have already covered will be reviewed in more detail and, where appropriate, source code will be made available.

I won't be getting into a lot of detail this month. I want to first share a few of my experiences in bringing CP/M Plus up for the first time and my initial reaction to using it.

Implementation

I chose to rewrite my BIOS for CP/M Plus (V3.0) from the ground up rather than attempt to adapt my V2.2 BIOS code. This was the most desirable approach for me because of a few specific changes that I had made to enhance V2.2 which would serve no useful purpose under V 3.0 and could potentially cause unnecessary problems. Starting out with a clean slate also allowed me to implement many of my wish list items without being concerned about their effect on other sections of an already working system.

Before starting any work I decided to write a complete *banked* BIOS that could be used as *nonbanked* simply by changing one IF statement in each program module. I also wanted to keep all related device-dependent logic in separate modules for ease of debugging. The sample routines provided by Digital Research follow this same scheme and were very beneficial not only as a guideline, but also as a basis to work from.

Since I am an advocate of Zilog mnemonics, my first step was to convert the sample routines from Intel mnemonics to the Zilog standard with XLATE2. Each module was then edited for aesthetics. This step is not necessary if you don't care for Zilog mnemonics or if you want to use the sample routines as written. There is no real benefit to using one method over the other. It's simply a matter of personal choice. No matter how you go about putting your system together, I do strongly suggest that you at least use the sample routines as a guideline. Otherwise, you will waste a lot of valuable time in unnecessary coding and debugging.

I was then ready to begin the real task of writing, or rewriting in some cases, all the routines to interface V 3.0 to my hardware configuration. I was surprised when I finished after only three days because I had originally planned on spending about a week writing code. I think a week is a more realistic time estimate unless you are able to do as I did and shut yourself away to avoid any disruptions.

I was fortunate in being able to salvage a major portion of my V2.2 disk controller code which accounts for at least half of my complete system. Adding the Extended Disk Parameter Blocks and related control structures was an easy task that required very little new coding. The time-consuming parts were the new modules: character I/O module (CHARIO), boot routine (BOOT), bank selection and memory-to-memory move routine (MOVE). To complete my suite of modules I used DR's main module, BIOSKRNL, almost verbatim. Everything fell into place quite easily and looked like it should work without a considerable debugging effort. I wouldn't realize how complex the placement of code between common and banked memory would be until I started testing in a banked environment. But this is another subject altogether that I will talk about a little later.

It was my intention from the beginning to limit the complexity of my first test system as much as possible in hopes that my debugging effort would also be simplified. When generating my first nonbanked system with GENCPM, I answered "no" to as many questions as I could and limited buffer allocations to one each for both data and directory. Fortunately, luck was with me and on my third attempt I had CP/M Plus operational.

After running a few test programs I was confident enough to begin generating more test systems with a few more features enabled than the one before it. Everything was holding together better than I thought it would until I started testing my first *banked* system.

I was sure that my bank switching hardware was stable because I had written a diagnostic program to verify its integrity and had been using it daily with V 2.2 without any problems. Using SID to assist in debugging a banked system is no help at all because once a bank is switched out SID will typically crash or cause its own share of strange errors. What is needed is a program that does not depend on the lower map of memory or any BDOS functions to operate. As it turns out, repeatedly changing the code, relink-editing and generating, and testing were practically the only way I had to get my banked version up.

Even though I had a banked version operational on the first day, it was prone to sporadic failures. Finally, after two weeks and the eradication of a few bugs, I was completely operational with a stable system. During this time I learned a lot about how CP/M Plus works by reading and rereading the manuals and just poking around in an attempt to find out what was going wrong.

Based on my experience, I cannot overly stress the importance of studying the manuals that come with the CP/M Plus system before charging off to bring it up. Even though the manuals have been drastically improved, a number of very important tricks are well hidden in the text. In addition, CP/M Plus is a completely new system with very little resemblance to its predecessor.

User Report

Using CP/M Plus is a joy. The extended command editing feature available on banked systems saves a nontypist, like myself, a lot of time. If a keying error is made in the middle of a command line it is no longer necessary to backspace and erase to the incorrect character and then retype the remainder of the line. With CP/M Plus you simply type CTL-A to nondestructively backspace over each character in the command until you reach the error. You then have the choice of using either CTL-G to delete one or more characters at the cursor position or of making insertions by typing the new text. If the command line is lengthy you can also skip to the right or left end of the string with a single keystroke.

The entire command line can also be redisplayed even after return has been hit and a transient program has executed. One of my most common errors is to misspell file names which causes difficulties at the retrieval stage. Provided no other keyboard entries have been made, hitting CTL-W will redisplay the last command line entered and allow further editing. This one feature alone has saved me from many frustrating moments at the keyboard, especially when I am in a hurry. One other very practical use for this facility is when the same program must be run against a number of files. Rather than setting up a submit procedure you can now easily edit the same command line as many times as necessary.

And last but not least, multiple commands can be entered on one line by separating them with an exclamation mark.

I couldn't be happier with the keyboard editing features except for one gaping hole. There is no way to redefine the control sequences assigned to individual functions. I would have thought that this medieval practice, akin to the Chinese water torture, would have been outlawed in a new development effort.

Submit file processing is as automatic as you want it to be. After proper specification of the search path with the SETDEF program, a search is made for the .COM file specified in the command line. If it isn't found the search is automatically continued, only this time a file type of .SUB is used. When the desired file is found, SUBMIT.COM is loaded for execution and the specified file is used as input.

Program input can be imbedded in the submit file by preceding each input line with a less-than sign. In the case of an errant program that tries to read program data after the file is exhausted, further input requests are directed to the keyboard. In addition, system commands are protected from being read as program data.

Limited conditional submit file processing can be done by prefixing system commands with a colon. This indicator tells the CCP to check the system error code for a nonzero value, and if true, skip the command.

And in my estimation the most useful new feature is nested submit files. You are no longer prohibited from calling submit procedures from other submit procedures. This is especially useful when doing any kind of batch processing.

Supercharged

Quite an uproar was heard when the truth was known about the LRU buffering logic of CP/M Plus. The most heated public discussion took place in *DDJ* a few months ago. Dave Cortesi took the time to uncover the LRU buffering logic, only to find that for sequential files, there wasn't any. The fact is, LRU buffering is disabled for sequential file processing.

Fortunately, Digital Research is not a company to be kicked for very long before rectifying a situation. The application note at the end of this column changes the LRU buffering logic to give equal priority to buffer usage regardless of how a file is accessed. Exactly what this change means in terms of program runtime is unknown as yet because I haven't run any benchmarks to make a comparison between a before and after system, although from outward appearance I would think that a substantial savings is in order.

Tips

When coding the physical device names in the character table, make sure they are entered in upper case. The DEVICE transient program converts all keyboard input to upper case before searching for a match in the character table.

Application Note

CP/M Plus V3.0, Patch 13, 5/1/83
BDOS Patch 02

Copyright © 1983 by Digital Research Inc. CP/M Plus and SID are trademarks of Digital Research Inc. Compiled September 1983. Printed with permission of Digital Re-

search (but see the Editor's Note in the listing).

Products and Serial Numbers that Require Updating: CP/M Plus V3.0, serial numbers 2-000-00001 through 2-000-XXXXX

Program: RESBDOS3.SPR, BNKBDOS3.SPR, BDOS3.SPR

Error Description:

These patches do the following:

- clear the multiple command buffer if CTRL-C is encountered.
- change the LRU algorithm that manages data DCBs (banked systems only).
- correct the problem that occurs if a BIOS READ ERROR is encountered during login on a permanent drive.
- correct errors that oc·ur if directory write operations are performed to disks set to Read Only (R/O).
- correct random record I/O error that occurs when the random record number is greater than 3F000h.

Patch Procedures:

Make a backup copy of RESBDOS3.SPR, BNKBDOS3.SPR, and BDOS3.SPR before making any changes. The program SID is required to make the changes. The changes are made by the sequence of commands in the Listing (below). ■■J

Reader Ballot
Vote for your favorite feature/article.
Circle Reader Service **No. 191.**

CP/M Exchange (Text begins on page 14)

```
15C>REN RESBDOS3.SAV=RESBDOS3.SPR
15C>SID RESBDOS3.SAV
C:SID     COM
CP/M 3 SID - Version 3.0
NEXT MSZE  PC  END
0900 0900 0100 CCFF
#S41A
041A F8 FC
041B C8 .
#S797
0797 03 07
0798 06 .
#WRESBDOS3.SPR
0010h record(s) written.
#G0

15C>REN BNKBDOS3.SAV=BNKBDOS3.SPR
15C>SID BNKBDOS3.SAV
C:SID     COM
CP/M 3 SID - Version 3.0
NEXT MSZE  PC  END
3600 3600 0100 CCFF
#A529
0529  CALL 2D24
052C  .
#S6C6
06C6 08 10
06C7 C2 .
#SB92
0B92 A0 97
0B93 09 .
#S13B1
13B1 C0 00
13B2 CD .
#A15ED
15ED  CALL 0EEB
15F0  JNZ 13FF
15F3  LHLD 2871
15F6  CMP M
15F7  NOP
15F8  NOP
15F9  JZ 13FF
15FC  JMP 2D40
15FF  CALL 1252
1602  CALL 1258
1605  .
#A1640
1640  JZ 2D6A
1643  .
#SW19DB
19DB 14F0 2D83
19DD CDC8 .
#A1B16
1B16  CALL 2D76
1B19  .
#SW1BA4
1BA4 14F0 2D83
1BA6 B9C2 .
#SW24F6
24F6 1E90 2D7D
24F8 36C3 .
#SW25F0
25F0 1E90 2D7D
25F2 52C3 .
#SW261B
261B 1E90 2D7D
261D A4CD .
#SW2704
2704 2510 2D3A
2706 20CD .
#A2C92
2C92  CALL 2D30
2C95  .
#A2CE5
2CE5  LDA 2D39
2CE8  ORA A
2CE9  NOP
2CEA  .
#S2D05
2D05 F8 F6
2D06 28 .
#S2D0D
2D0D F6 F8
2D0E 28 .
#A2F24
2F24  LXI H,0
```

EDITOR'S NOTE:

This is still a preliminary version of patch 13, not an official patch. It may contain some errors (though we have not heard of any yet). It is thus not endorsed for application by Digital Research. It is currently under review at DRI, and we had expected it to be approved by the time we went to press. Unfortunately, word of last-minute delays came too close to press time for us to withhold the listing. Thus we include this caveat. DRI will keep us updated, and we will let you know of its disposition and any changes that might be made.

(Continued on page 18)

PC NETWORK

Buy Hardware/Software at Wholesale, And Save On Software Rentals, As A NETWORK™ Member Only!

Save hundreds of dollars when you buy DIRECT from America's Number 1 Computer Buying Service at just 8% above DEALER WHOLESALE PRICES, plus shipping.

Members receive The Personal Computer NETWORK's Giant Catalog featuring thousands of products and the lowest prices on the widest selection of computer software and hardware in the nation!

RENT BEFORE YOU BUY — Members are eligible to join The NETWORK's Business and Game Software Rental Libraries for a much smaller fee than other software rental services. And The NETWORK's rental charges are far less — just 20%-25% of the Member WHOLESALE PRICE!

Join The NETWORK today for as low as $8 for one year (or $15 for two years) and receive all these exclusive, money-saving benefits:

1. **REAL BUYING CLOUT**—Buy at just 8% above DEALER WHOLESALE PRICES, plus shipping. (On credit card orders there is a 3% service charge.)
2. **CONVENIENT SHOP-AT-HOME CATALOG**
3. **KNOWLEDGEABLE SERVICE CONSULTANTS**
4. **FULLY INSURED FAST HOME DELIVERY**
5. **OPTIONAL BUSINESS SOFTWARE RENTAL LIBRARY**—Members join for just $30 per year in addition to the basic membership fee. **Rent business software at just 20%-25% of The NETWORK's low prices** for a 7-day period (plus a 3-day grace period for return shipping). **100% of your rental fee applies towards purchase.**
6. **OPTIONAL GAME SOFTWARE RENTAL LIBRARY**—Members join for just $10 per year in addition to the basic membership fee. All the same conditions apply as for benefit five
7. **SPECIAL SAVINGS BULLETINS** — Save even more on limited-quantity merchandise!
8. **DISCOUNT COMPUTER BOOK LIBRARY**—Save up to 50%!
9. **MEMBERSHIP SATISFACTION GUARANTEE**—If you are not satisfied, notify us within 30 days to receive a **100% money-back refund.**
10. **PRODUCT SATISFACTION GUARANTEE**—If you are not satisfied with any hardware, return it within 15 days for a **100% money-back refund.**

All items subject to availability, prices subject to change without notice

Copyright ©1983. PC NETWORK INC

Low prices, fast home delivery and two software rental libraries are only the beginning! The NETWORK is your source for everything from memory chips to mainframes—and it's all just 8% above wholesale, plus shipping.

HARDWARE

Monitors (color and monochrome)
Printers
Complete Systems
Disk Drives (full/half height, add-on/add-in)
Multi-Function Boards
Graphics Boards
Modems
Local Area Networks
Memory Chips (all speeds available)
S-100 Components

SOFTWARE (rent or buy!)

Business
Recreational
Compilers
Word Processors
Utilities
Scientific
Data Bases
Educational
Graphics
CP/M-MS/DOS

SUPPLIES & ACCESSORIES

Blank Diskettes (all formats)
Paper Stock
Ribbons
Print Wheels
Cables
And More!

Choose hardware and software from hundreds of manufacturers, including:

- Altos
- Amdek
- Apparat
- AST
- Ashton-Tate
- Atari
- CDC
- Coleco
- Columbia
- Commodore
- Corona
- Cromemco
- DEC
- Digital Research
- Eagle
- Epson
- Franklin
- Hayes
- IBM
- IUS
- Lotus
- MicroPro
- Microsoft
- Morrow
- Motorola
- NEC
- Northstar
- Okidata
- Peach Tree
- Princeton
- Quadram
- Sanyo
- Sierra On-Line
- Software Publishing
- Softword
- Tandon
- Tecmar
- Texas Instruments
- Toshiba
- Visicorp
- Zenith

SAVE ON UPGRADES!

NETWORK SPECIAL

Disk Drive	Retail	Wholesale
Two Tandon TM 50-2 Half Height DS/DD	$574/ pair	$350/ pair*
	Retail	Wholesale
64K Chips	$11 each	$5.10 ea.*

*NETWORK Members pay just 8% over the wholesale price, plus shipping

SPECIAL V.I.P. MEMBERSHIPS

Available at $15 per year or $25 for two years. V.I.P. Members receive additional benefits:

1. **BOTH SOFTWARE RENTAL LIBRARIES FOR $35** in addition to the V.I.P. membership fee.
2. **EXTENDED 10-DAY SOFTWARE RENTAL PRIVILEGES** (plus 3 days for return shipping).
3. **ADVANCE NOTIFICATION OF SPECIAL SAVINGS BULLETINS.**

CALL TOLL FREE

1-800-621-S-A-V-E

In Illinois call (312) 372-4488

Your Membership Validation Number: D463

You can validate your membership number and, if you wish, place your first money-saving order over the phone by using your VISA, MASTERCARD or AMERICAN EXPRESS. Our knowledgeable service consultants are on duty Mon-Fri 8 AM to 7 PM. Sat 9 AM to 5 PM CST.

Call now...Join the PC NETWORK and start saving today!

Circle **no. 55** on reader service card.

THE NETWORK • MEMBERSHIP APPLICATION

YES! Please enroll me as a member in The NETWORK™ and rush my catalog featuring thousands of computer hardware and software products, all at just 8% above DEALER WHOLESALE PRICES. I will also receive all the other exclusive, money-saving services available to Members.

I am under no obligation to buy anything. My complete satisfaction is guaranteed.

Please check (✓) all boxes that apply:

Basic Membership

□ One-year membership for $8
□ Two-year membership for $15 (SAVE $1)
□ Business Software Rental Library for $30 add'l. per year—members only
□ Games Software Rental Library for $10 add'l. per year—members only

Special V.I.P. Membership

□ One-year membership for $15 871
□ Two-year membership for $25 (SAVE $5)
□ BOTH Business and Game Software Rental Libraries for $35 add'l. per year—V.I.P. members only

□ Bill my credit card: □ VISA
□ MasterCard □ American Express Exp. Date ______ mo.year

Account Number:
□□□□□□□□□□□□□□□□

□ Check or money order enclosed for $________

Name________

Address________ Apt. No.____

City________ State______ ZIP______

Telephone (______)______

My computer(s) is: □ IBM PC □ Apple II □ TRS-80
□ Atari □ Commodore Other________

Signature ✓________
(Signature required to validate membership)

CP/M Exchange

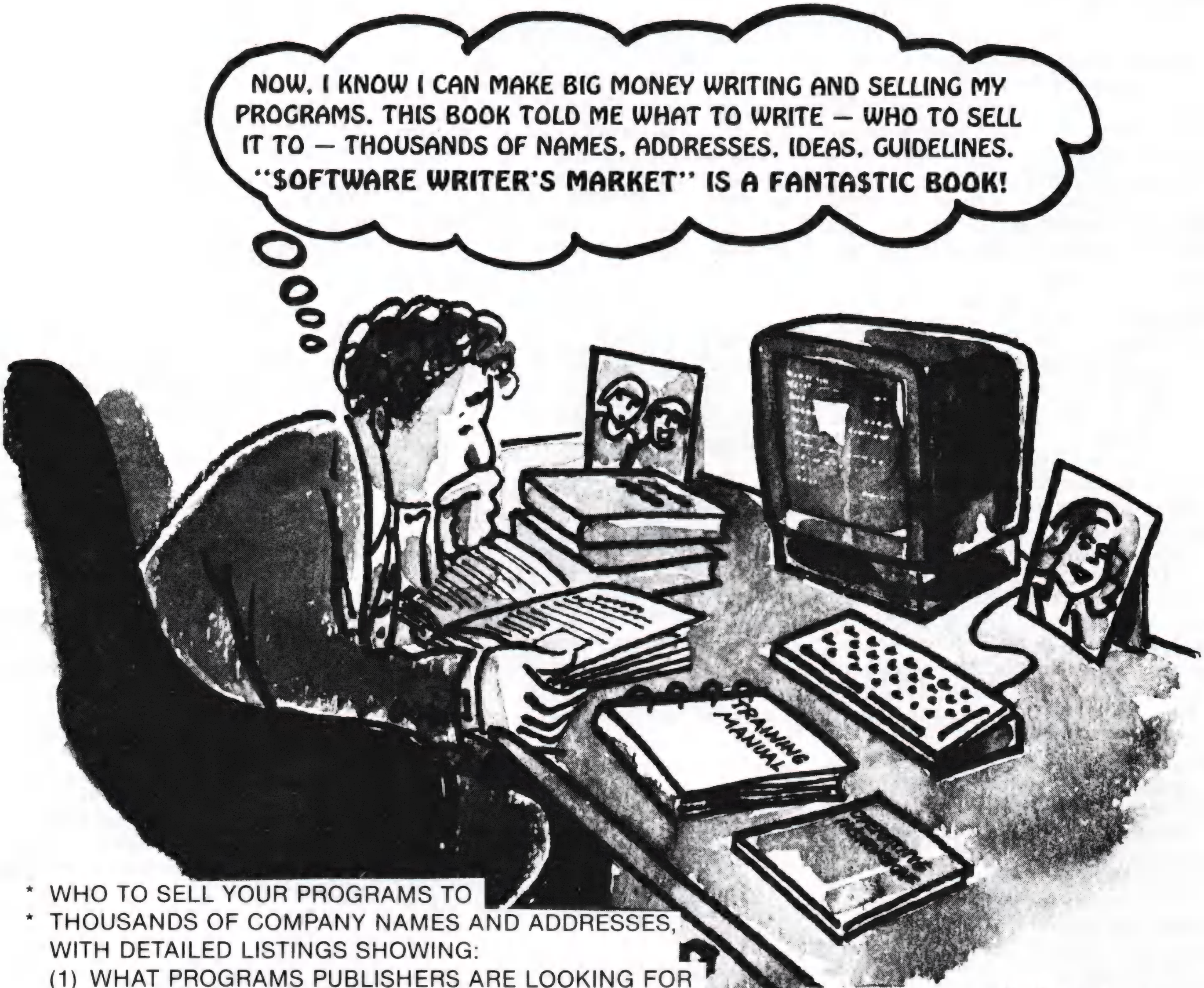
(Listing continued, text begins on page 14)

```
2F27  SHLD FBBA
2F2A  SHLD FBB1
2F2D  DCX H
2F2E  DCX H
2F2F  RET
2F3Ø  SHLD 28F4
2F33  SUI 3
2F35  STA 2D39
2F38  RET
2F39  NOP
2F3A  CALL 2D43
2F3D  JMP 2513
2F4Ø  CALL 1377
2F43  LHLD 287B
2F46  MOV A,L
2F47  ANA H
2F48  INR A
2F49  RZ
2F4A  MOV E,M
2F4B  INX H
2F4C  MOV D,M
2F4D  MOV A,D
2F4E  ORA E
2F4F  RZ
2F5Ø  LXI H,28AA
2F53  LDAX D
2F54  CMP M
2F55  JNZ 2D63
2F58  LXI H,4
2F5B  DAD D
2F5C  MVI A,FF
2F5E  CMP M
2F5F  JNZ 2D63
2F62  STAX D
2F63  LXI H,D
2F66  DAD D
2F67  JMP 2D4A
2F6A  CALL 1377
2F6D  LHLD 2871
2F7Ø  MOV A,M
2F71  ORA A
2F72  RNZ
2F73  MVI M,2
2F75  RET
2F76  CALL 1139
2F79  LXI H,FD17
2F7C  RET
2F7D  CALL 1E9Ø
2F8Ø  JMP 1139
2F83  CALL 1162
2F86  JMP 14FØ
2F89  .
#S3Ø65
3Ø65 82 92
3Ø66 4Ø .
```

```
#S327F
327F 9Ø 12
328Ø 49 .
#S35A5
35A5 ØØ 48
35A6 ØØ 21
35A7 ØØ Ø9
35A8 ØØ 24
35A9 ØØ ØØ
35AA ØØ 21
35AB ØØ ØØ
35AC ØØ 4Ø
35AD ØØ 49
35AE ØØ ØØ
35AF ØØ 91
35BØ ØØ 24
35B1 ØØ 8Ø
35B2 ØØ .
#WBNKBDOS3.SPR
ØØ6Ah record(s) written.
#GØ

15C>REN BDOS3.SAV=BDOS3.
    SPR
15C>SID BDOS3.SAV
C:SID       COM
CP/M 3 SID - Version 3.Ø
NEXT MSZE  PC  END
278Ø 278Ø Ø1ØØ CCFF
#AØ4C2
Ø4C2  CALL 1E25
Ø4C5  .
#S642
Ø642 Ø8 1Ø
Ø643 C2 .
#S846
Ø846 54 4B
Ø847 Ø6 .
#SD6F
ØD6F CØ ØØ
ØD7Ø CD .
#SW12A2
12A2 ØE86 1E3E
12A4 CDC8 .
#A13DD
13DD  CALL 1E31
13EØ  .
#SW1463
1463 ØE86 1E3E
1465 78C2 .
#SW1B29
1B29 16A5 1E38
1B2B 63C3 .
#SW1B8A
1B8A 16A5 1E38
```

```
1B8C 3AC3 .
#SW1BB5
1BB5 16A5 1E38
1BB7 71CD .
#A2Ø25
2Ø25  LXI H,Ø
2Ø28  SHLD 1EBA
2Ø2B  SHLD 1EB1
2Ø2E  DCX H
2Ø2F  DCX H
2Ø3Ø  RET
2Ø31  CALL ØAF7
2Ø34  LXI H,1CDF
2Ø37  RET
2Ø38  CALL 16A5
2Ø3B  JMP ØAF7
2Ø3E  CALL ØB2Ø
2Ø41  JMP ØE86
2Ø44  .
#S2Ø97
2Ø97 Ø2 Ø6
2Ø98 Ø6 .
#S233F
233F 41 49
234Ø 2Ø .
#S26AC
26AC ØØ 24
26AD ØØ 12
26AE ØØ 24
26AF ØØ 9Ø
26BØ ØØ .
#WBDOS3.SPR
ØØ4Dh record(s) written.
#GØ
```

End Listing


NOW, I KNOW I CAN MAKE BIG MONEY WRITING AND SELLING MY PROGRAMS. THIS BOOK TOLD ME WHAT TO WRITE — WHO TO SELL IT TO — THOUSANDS OF NAMES, ADDRESSES, IDEAS, GUIDELINES. "$OFTWARE WRITER'S MARKET" IS A FANTA$TIC BOOK!
TRAINING MANUAL

Micro to Mainframe Connection

by Keith Coye and
Alvin Grossman

Keith A. Coye, Consulting Services Corporation, 7214 Via Maria, San Jose, CA 95139.

Alvin Grossman, San Mateo Co. Office of Education, 333 Main Street, Redwood City, CA 94073.

Artwork: Joe Murray.

Long ago and far away there lived in the mind of Man a single large machine that could be all things to all people: the Main Frame Computer. To further Man's purposes the Main Frame Computer grew larger and larger and more complex. To tend its every need a priesthood of Programmers and Analysts evolved. They and only they were permitted to talk to the large machine. As in all priesthoods and cults, they spoke in strange tongues that only the large machine understood. All others had great difficulty communicating with the large machine and had to rely on the priesthood so that the machine would be bountiful and smile upon them and do their work.

Strangely, the people also found it difficult to talk to the priests, for the priests were more interested in talking to the large machine than to the people: a perplexing problem and one that did not seem to be amenable to an easy solution. Then into this inhospitable land of Babel rode a knight on a pure white charger: the Microcomputer!

All at once the people had within their grasp the world the priesthood had been promising, lo, these many years. They no longer needed the priesthood. Temples were smashed, large machines were shunned, a new day had arrived. All the people needed was access to a microcomputer and the proper software. Rejoice, the people were no longer beholden to others; they were in control of their own destinies.

Were it all so easy. Alas, the lack of software, the emphasis on Games, the difficult-to-understand manuals left the people wringing their hands. Perhaps they needed the priesthood after all. But in spite of the many early woes, software got better, people began to talk in fewer tongues, and the microcomputers got bigger and bigger. The Programmers soon found that they needed a means of exchanging information among the various systems, but each machine spoke in its own tongue. What was needed was a common language for transferring data among the various machines.

Here we leave our fairy tale and return to reality, although we very much hope that, like a fairy tale, there will be a happy ending. Many of us felt that the microcomputer was not a threat to our "empires" but instead was a godsend that would extend the boundaries of computing and hasten the arrival of true distributed processing. Although we were called foolhardy by many of our contemporaries in the priesthood, it was our perception that the microcomputer would usher in the day of popular computing and would effectively allow a melding of computing power and communications. It would put computing power into the hands of the users and, with it, the decisions on how they would want to use it.

"IN THE BEGINNING"

Planning the Connection

At the San Mateo County Office of Education in the heart of the Silicon Valley, we began to plan our efforts way back in 1980 (it seems like eons ago). At that time our Three Year Master Plan stated:

"We have taken the view that micros are not antithetical to our current environment. In fact, through the development of this plan, the use of a variety of computers will result in an environment that will be more user oriented and make a full spectrum of computational power available to all users. We will move from being an organization with primarily an administrative focus to one that will provide the necessary conduits for delivery of a wide variety of systems.

"In addressing this new phenomenon it must be remembered that this is a new technology and not simply a cheaper or necessarily better way of doing things. What in fact is happening is that the nature of user interaction with computing has changed and the range of computing applications has expanded dramatically. The incorporation of micros into our 'systems' is especially fortuitous at this time. It will allow us to be able to off-load minor and special-purpose functions and to expand our service offerings. It will also prolong the life of the current large computer (DECsystem-10). In addition to what has been traditional computer center functions, an expanded role will be added, that of more far flung telecommunications to expand the availability of computing to all districts and schools in our service area and to make their access easier.

"This plan proposes a means whereby we will become essentially a network, from the smallest micro to the major mainframe at our headquarters, with all users accessing and being able to obtain any files they want and need in a manner most advantageous to them."

Our goals at that time were explicitly stated as follows:

1. Move toward a marriage of micros/ mainframes
2. Design an interface between the two machines to facilitate the easy communication and downline loading of data despite differences in operating systems, disk formats, and text format conventions

Discover the wizardry of MAGIC

MAchine Generated Integrated Code©

MAGIC/MPS™—A revolution in program development. MAGIC supersedes COBOL, PASCAL, C, BASIC and other application languages. MAGIC enables you to produce good clean assembly language machine code—and plenty of it.

MAGIC through its various compilers allows the use of the same source to derive multiple object codes for a wide variety of systems.

MAGIC makes a run-time package a thing of the past. MAGIC lets you address diverse system and application needs in significantly less time.

For more information, send coupon or
write Data Management Assoc., Inc.,
P.O. Box 4340, Wilmington DE 19807.
Phone 302/655-8986.

MAGIC MAKES PROGRAMMING EXCITING

MAGIC is a language written for programmers by programmers.

MPS/80™ for CP/M™ systems
MPS/86™ for CP/M-86™ and MS™-DOS systems

SHOW ME SOME MAGIC!

Mail coupon to Data Management Assoc., Inc., PO Box 4340, Wilmington, Delaware 19807, or call 302/655-8986.

My system is________________________.

Send: □ More information
□ MAGIC/MPS™ Manual @ $50 ea.
□ MPS/80™ @ $795 ea.
□ MPS/86™ @ $995 ea.
□ Check enclosed □ Send C.O.D.

Name________________________

Title________________________

Company________________________

Address________________________

City/State/Zip________________________

Phone(   )____________Best time to call________

SOFTCON

LET US SHOW YOU SOME MAGIC AT BOOTH L6019

CP/M and CP/M-86 are registered trademarks of Digital Research, Inc.; MS-DOS is a registered trademark of Microsoft Corporation.

Circle **no. 21** on reader service card.

3. Design an easy-to-use system to allow "packages" of data to be downline loaded and then manipulated by a micro data base manager
4. Create a resource group to act as a primary "help" group to teach users how best to access and use their data bases on the mainframe
5. Design approaches that would allow the mainframe to be a primary storage and switching network and allow a great deal of the computing to be at the local level on micros

Initial Prototyping

We decided to prepare a prototype implementation connecting the DECsystem-10 at the San Mateo Center with a number of Radio Shack Model IIIs. They provided a large population of systems and were Z80 based. The choice of the Radio Shack Model III was based on the large population of this type of system in our area and the ease with which we could extend our software to other similar 8080-based systems. We felt that this would give us a good perspective on the challenges of connecting a network of non-CP/M systems to the DECsystem-10.

One of the first hurdles we faced was the choice of programming language for this application. We had both microcomputer- and mainframe-level coding to do and wanted to consider both ease of implementation and ease of maintenance. We evaluated different languages as follows:

1. DECsystem-10 assembly language – a good candidate from the perspective of its close relationship with the DECsystem-10 operating system and the ease of communication with operating system internals, but a liability since we could not transport the code to another mainframe or microcomputer
2. Fortran – a good candidate since Fortran runs on most systems, both mainframe and micro, but the liabilities included a lack of good string handling and isolation from the critical terminal and monitor internals required for this type of application
3. COBOL – again, a good candidate since COBOL runs on most systems, although it is weak in the area of efficiency and in the ability to control modem and communications areas

Further investigation led us to evaluate an emerging language, C, as a candidate. Lo and behold, it had the structure of a very high level language, nearly the efficiency of assembly code, and best of all was reputed to be highly portable. Off we went to choose a good C compiler

Evaluation of C Compilers

Very few C compilers were available three years ago, so our field of choice was rather limited. Most of the C compilers for the DECsystem-10 mainframe were implemented by universities or schools. The manufacturer had no product, so we turned to the educational area for help.

Several versions of C were located at the University of Utah and Tufts University in Boston. The version at the University of Utah was in development and had licensing restrictions that prohibited its use as a future commercial product. The version created by David Krumme in the Mathematics Department at Tufts was a good compiler although very newly developed.

On the microcomputer side, the BDS C compiler was one of the few that had any proven track record for the microcomputer community. Despite its nonstandard library and the difficulty of linking in assembly code, we chose it as our initial compiler.

We quickly discovered that the non-CP/M environment required a special library and support for the TRSDOS operating environment. Our strategy was to compile the main application code on a Z80 development machine under CP/M, unload the "com" file into an Intel hex file with an origin suitable for the TRS80, transfer the hex file to the TRS80, and create a special loader on the TRS80 to create an executable file on the TRS80. Needless to say, this was a complex, time-consuming, and often frustrating process. The debugging time was astronomical! Each bug that was discovered required a complete recompilation, unloading, transfer to the TRS80, and reloading to try again.

It soon became obvious that the process of transporting the code from one micro environment to another had to be improved. We decided that we must find a C compiler that ran on more than Z80-based systems, had a standard Unix-like library, and allowed easy integration of assembly code using standard linking loaders. The C code needed to be transportable to other systems with an absolute minimum of source code change.

The Final Selection

Enter Aztec C from Manx Software!

Early testing showed that, although the compilation and execution speed was slightly slower than the BDS C compiler, the claims for portability, assembly code integration, and standard library functions were true. We immediately converted the BDS C source code to the more standard Unix-like syntax used by Aztec C.

The communication software, now called XPRESS, came to life, and we found that the code written in Aztec C was portable not only to other microcomputers but also to the DECsystem-10 mainframe. Our strategy was to create common code segments for all machines and to design "personality modules" to adapt the specific operating system, modem, and communication requirements. The TRS80 version was a success and soon requests came in for Televideo, Apple, CompuPro, Radio Shack Model 2, and Radio Shack Model 16.

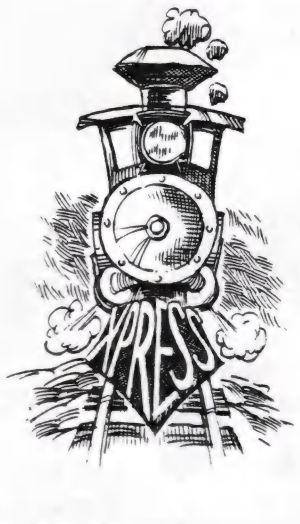
Then the IBM PC was announced and the race was on to get the C compiler up and running on the PC. Manx had not yet released its version for the PC so an alternative vendor, Computer Innovations, was selected as an interim candidate. The compiler was good in all respects, but it did require some modification since it did not adhere to some of the I/O syntax conventions used by Aztec C. Shortly thereafter, Manx released its Aztec C for the PC and we decided to continue with it for the PC development.

Software Design

The internal software design of XPRESS is described below. It may be of interest to those readers who enjoy the more technical side of the communications arena.

XPRESS is designed to allow a common memory area to be used as required for terminal communications, file transfer, and general-purpose temporary working space during directory sorts and file operations. The common area is managed internally within XPRESS, and the user never needs to worry about which section of the common area is being used at any given time for a specific function.

A portion of the common memory area, the text buffer, is used to receive all terminal transmissions from the remote computer. A series of pointers to text within the buffer allows the user to review terminal activity even after it has scrolled off the screen; automatically capture text for use with systems that do not support protocol file transfer; and provide buffered printing of the text, even if the printer runs slower than the communications line.



XPRESS allows these three functions to operate in the text buffer without error or interference with each other. The text buffer typically takes up as much memory as is available after the program is loaded. When the text buffer is being used only as a review buffer, then the buffer simply wraps around to the beginning as soon as it fills; the last buffer of information is always available for review.

When the text buffer is being used for nonprotocol file transfer, the Capture pointer keeps track of the contents of the buffer. As soon as the buffer is about 95 percent full, XPRESS stops the transmission of additional characters (using XOFF) and saves the buffer to a specified file before allowing the file transmission to restart.

When the text buffer is being used as a printer buffer, XPRESS sends characters to the printer while buffering incoming characters. If the printer is either busy or off line, XPRESS continues to fill the text buffer and resumes printing when the printer is again available. If the printer is so slow that the text buffer gets nearly full, XPRESS requests the remote system to suspend transmission by sending the appropriate character, usually XOFF (decimal 19).

Terminal Emulation

XPRESS emulates a DEC VT100 terminal in those modes possible on each specific microcomputer. These may include cursor position, cursor up, down, forward, and backward, device status report, cursor position report, save cursor position, restore cursor position, home erase to end of screen, erase to end of current line, set graphics rendition, set mode, reset mode, and keyboard key reassignment as specified in the ANSII terminal specification. The VT100 was chosen based on the large volume of software written for that terminal and its popularity in the mainframe marketplace.

File Transfer Protocols

The CRC protocol used in this implementation is an extension of that proposed by Ward Christensen in his Modem series programs. It is extended to allow the transmission of batches of files and to use the CCITT 16-bit CRC algorithm. The CRC protocol was chosen for its superior error recovery and also because a large number of microcomputer bulletin board systems already use it for file transfer.

The algorithm (x16 + x12 + x5 + 1) is specified as providing detection of all single- and double-bit errors, all errors with an odd number of error bits, all burst errors of length 16 or less, 99.9969 percent of all 17-bit error bursts, *and* 99.9984 percent of all possible longer error bursts. (See *Computer Networks*, by Andrew S. Tanenbaum, Prentice-Hall, 1981.) The CRC is set to zero at the start of each packet received and then accumulated for each data byte. At the end of the packet the CRC is updated with the MSB of the CRC and then the LSB of the CRC. The CRC should be zero at this time if no error occurred in transmission.

char 0	Start of Header (SOH-decimal 1)
char 1	Packet number modulo 64
char 2	Complement of packet number modulo 64
char 3-130	8-bit data bytes with no encoding of any kind
char 131	Most significant 8 bits of 16-bit CRC per CCITT-16
char 132	Least significant 8 bits of 16-bit CRC per CCITT-16

Table 1.

Data are transferred between systems by taking a sector of information from the disk and surrounding it with identifying information as well as information needed for error recovery. This block of data and administrative information is often called a packet. The actual packet format is as shown in Table 1 (above).

The ACK character (6 decimal) is used as a positive acknowledgement, and the NAK character (21 decimal) is used as the negative acknowledgement. The receiver initiates the connection by sending a capital "C" character (67 decimal) until it receives the first packet. This "C" character signals the transmitting machine to use the CRC protocol; it then uses the ACK and NAK as detailed above to confirm each packet. Packets refused by the receiver are retransmitted up to five times before giving up. Fatal errors are communicated by sending the CAN character (24 decimal) three times to the opposite system.

The file name packet is a special packet differing from a standard packet only in that it is null filled after the file name in the data area. A data packet filled with all nulls indicates the end of the batch of files being transmitted.

Note that the CRC protocol requires the use of all eight data bits and will send ACK, NAK, CONTROL characters, and characters with values in excess of 128 decimal during the normal transfer of ASCII and binary information. This protocol, therefore, cannot be used in systems that restrict the use of 8-bit characters in any way.

The XNET protocol is a fully packetized protocol useful in cases where the line will not allow all characters to pass freely. It is a proprietary protocol that maps characters into the dense graphics printable character set, thus allowing transmission over restricted lines. Packet size is restricted on configurations using the DECsystem-20 due to the inability of the DECsystem-20 to handle packets of data sent in bursts of greater than 94 characters. This protocol has receiver-defined parameters, including packet size and pre-packet and post-packet terminators, making it suitable on a large number of remote systems.

Related Software

Teachers today have a huge selection of software to choose from and often may lack the time to personally try out each package. We felt that a means to quickly select and review important facts about each available package was needed. A data base of software packages was developed, which provides the teacher with a means of obtaining a quick overview of all packages that meet specified selection criteria. The user may then request additional detailed information about any software package.

An example of how this software library application works might be helpful at this point. Let's suppose that a math teacher has an Apple computer and wishes to provide instruction to the students on how to find the area and volume of common shapes and solids. The teacher would use the Apple computer to communicate with the mainframe in terminal mode and would access the software library application program. Next, the teacher would request information on those software packages that run on an Apple, whose subject is math, grade level is sixth through ninth, and that are either available at no charge or commercially available.

The data base would be scanned, and information on the 15 programs in the data base that meet these criteria would be shown in summary format to the teacher. The teacher would then request specific information on three packages that seem to be most appropriate to the needs for the class. Additional information such as cost, system requirements, where the package could be obtained, and comments by other teachers on the package would then be shown to the teacher. The teacher could then elect either to download the desired public domain software directly using XPRESS or to contact the distributor for commercial packages.

Future Directions

It is apparent that school districts will continue to migrate toward micros for many applications and accordingly will make less use of the mainframe. Putting into practice one of our major goals, which is to put data processing control into the hands of the users, we created the PIPELINE system. PIPELINE is a user-oriented system that works with a multiplicity of microcomputers.

Using this system, users decide what data they need and how they want it parameterized for their special reporting or analysis needs. Then, using their micro as a dumb terminal, they access the mainframe and ask for PIPELINE. They are presented with a menu that asks them what kind of data they want. If, for example, they want personnel data, they are routed to a data dictionary that lists all of the personnel data elements on the system, each sequentially numbered with a one-line descriptor. Then they choose by number (such as 33, 96, and 101) which data elements are to be extracted.

If additional data from the financial system are needed to integrate with the personnel data, then the users ask for the financial menu and go through the same process to extract the needed financial data. When the users have extracted the defined data, they indicate the data should be downloaded to their micro. While interrogating the mainframe system for information, users may use Boolean Logic selectors such as greater than (>), less than (<), or equal to (=) in specifying what kinds and ranges of data they want (i.e., select teachers whose subject = math and who have > 5 years of experience).

Once this data has been downline loaded in a format that the users' micro can handle (using XPRESS as the vehicle), it then may be further manipulated using a local data base manager.

A complementary approach is the use of a software package that allows a large mainframe to run microcomputer software, including the popular CP/M operating system and the large quantity of inexpensive programs that run under it. With no additional hardware investment (such as the purchase of a micro), users may use their current terminals and operate them as if they were a fully configured microcomputer; at the same time they have the full power of the mainframe available.

The users could, of course, also use a micro to downline load any information to their floppy disks and run whatever programs they wished independently. The best of both worlds, this approach allows CP/M and CP/M-based programs, as well as stand-alone programs, to run on the DECsystems. Except for their speed and the fact that they are easily accessed remotely over telephone lines and networks, such programs run exactly as they do on a microcomputer.

Investigation is being done on techniques for data compression and encryption to meet the needs of high-volume data transfer with security. Among the techniques being explored are Hoffman coding, repetitive character encoding, use of private keys, and more efficient protocols.

The future is bright with promise for the connection of today's new series of microcomputer to the traditional commercial mainframes to provide the strength of the mainframe with the flexibility and local control of the microcomputer.

DDJ

Reader Ballot
Vote for your favorite feature/article.
Circle Reader Service **No. 192.**

COHERENT™ IS SUPERIOR TO UNIX*
AND IT'S AVAILABLE TODAY
ON THE IBM PC.

Mark Williams Company hasn't just taken a mini-computer operating system, like UNIX, and ported it to the PC. We wrote COHERENT ourselves. We were able to bring UNIX capability to the PC with the PC in mind, making it the most efficient personal computer work station available at an unbelievable price.

For the first time you get a multi-user, multitasking operating system on your IBM PC. Because COHERENT is UNIX-compatible, UNIX software will run on the PC under COHERENT.

The software system includes a C-compiler and over 100 utilities, all for $500. Similar environments cost thousands more.

COHERENT on the IBM PC requires a hard disk and 256K memory. It's available on the IBM XT, and Tecmar, Davong and Corvus hard disks.

Available now. For additional information, call or write,

Mark Williams Company
1430 West Wrightwood, Chicago, Illinois 60614
312/472-6659

Mark
Williams
Company

COHERENT is a trade mark of Mark Williams Company.
*UNIX is a trade mark of Bell Laboratories.

Circle **no. 85** on reader service card.

Communications Protocols:

Theory and Practice

How many of us have not had the frustration of trying to get some data from a friend, or some other micro or mainframe, only to discover that your machine's MICRO-COMM does not communicate very well with the mainframe's TSO or your friend's PC-LINK? By "communicate" I mean much more than just acting as a dumb terminal; I include full file transfer in both directions. The problem is an incompatibility of *protocols* between the machines and communications packages involved. In order to solve the problem, we need to decide just what a protocol is, how it works (or doesn't work), and what some typical ones look like.

Starting Simple: The XON/XOFF Protocol

Simply put, a protocol is an agreed-upon set of rules for two computers to communicate with each other, a "verbal handshake" between the computers, if you will. The simplest such protocol, and one with which most of us work every day, is the XON/XOFF (or DC1/DC3) protocol. Many terminals, such as the DEC VT-52 and Zenith Z-19, use this protocol to control the flow of data from the microcomputer. This illustrates one of the first goals of any protocol: To control the flow of data.

The XON/XOFF protocol uses the special characters XON (control-Q) and XOFF (control-S) to turn on and off the flow of data from the other end. CP/M uses control-S (XOFF) to stop the flow of data to the console, and any other key to restart it – a simple implementation of the XON/XOFF protocol. Printers often use this protocol to prevent their buffers from overflowing; a printer will receive data at 9600 baud until its buffer is about three-fourths full. At this point it will send an XOFF to the microcomputer, and the microcomputer will (hopefully) stop sending data. The printer will print for a while, until its buffer drops to only one-fourth full. The printer will then send out an XON, and the microcomputer will again send it data.

by Leslie Brooks

Leslie Brooks, Computing Center, Florida State University, Tallahassee, FL 32306.

In our hypothetical case everything worked exactly as it should. In the real world things are not so simple and problems often develop. For example, suppose that your cat pounces on the printer cable just as the printer sends out the XOFF. You live in Nevada and the air is very dry, so the cat has built up a static charge as it crept up on the cable. This static electricity discharges into the cable and the XOFF is drowned in noise. Your micro never sees the XOFF and continues shoving out data at 9600 baud. Hopefully the person who designed your printer's software anticipated this, and your printer sends out another XOFF when its buffer has only 100 bytes left. This illustrates the second goal of any good protocol: Error tolerance. Herein lies a *very* deep swamp!

Admittedly, most printer designers probably considered the case we looked at above, but how about a very similar case? Again, picture your cable getting a noise burst, but just as the XON goes out. Your microcomputer never sees the XON and doesn't transmit any more data. Most printers will allow this situation to continue until their ribbons rot and fall off! A clever designer might have the printer send out XON's every ten seconds or so if the buffer is empty and it is not receiving data. This is a third goal of a good protocol: To anticipate and prevent deadlock conditions. Not every protocol succeeds here, and the success or failure may often depend as much on the implementor as on the protocol.

Micro to Mainframe and Back Again

If you have stayed with me this far you can already understand most of the reasons why your microcomputer cannot talk to your friend's: they don't use the same protocol. This is also the problem with trying to communicate with many mainframes. In the real world, then, how do we surmount the problem? Well, there *are* some pretty standard protocols in wide use. For example, the CLINK protocol developed by Larry Hughes and first used in his CLINK communications package is also used in Crosstalk and in Larry's later package, MITE. (See the SEND/RECEIVE article in the August 1982 issue of *Dr. Dobb's* for some of Larry's earlier work.) The XMODEM protocol, developed by Ward Christensen, is in widespread use on many different machines and in many different implementations, and is in the public domain. Let's take a quick look at it.

XMODEM

The XMODEM protocol is a reasonably good protocol, widely used, and particularly suited to use on microcomputers. A typical transmission would look like

```
Sending     S  B  B  D  D  D ... D  D  C             E
Machine     O  L  L  A  A  A ... A  A  K             O
            H  K  K  T  T  T ... T  T  S             T
               #  #  A  A  A ... A  A  U
                  *                    M...      ...



Receiving                                    A
Machine                                      C
                                             K
```

**One's complement of the block number.*

This introduces some new terminology. SOH means Start Of Header and is a standard ASCII control character that is often used to indicate the start of a message. After the SOH is a block number, so that XMODEM can tell whether or not it has already seen this block or missed a block. The second block number is the one's complement of the first, which gives some assurance that XMODEM is not throwing away a block based on a corrupted block number. XMODEM defines that there will be 128 bytes of data following the block number. After the data comes the checksum, which is an error check code. The transmitting machine adds all of the data bytes together and transmits their sum (modulo 256) at the end of the block. The receiving machine also adds the data bytes and compares the sum it reaches to the one it received. If the two sums match, the data is assumed to be good and the receiving machine sends an ACKnowledge (as shown here); if the sums do not match it will send a NAK (Negative AcKnowledge). If the transmitting machine receives a NAK (or no response at all), it retransmits the last block sent. It continues retransmitting until it receives an ACK or gets too many errors and aborts. This gives some assurance that the data will arrive correctly and also avoids most deadlocks. (See "How Accurate Is Accurate," below, for further discussion of error detection.) As in our example, the transmitting machine sends an EOT (End

Of Transmission) to indicate that no more data remains after the last block has been acknowledged.

XMODEM, then, meets most of the requirements of a good protocol; why doesn't everyone use it? Well, some companies like to have their own communications packages, so they produce a proprietary protocol. Mainframes do not use it because they use other protocols, most of them much older than XMODEM. However, almost all CP/M bulletin board systems use it, and some packages (such as MITE) support it in addition to their own protocols. Even better (if you need to talk to minicomputers), there is a package called XCHANGE11 that implements the XMODEM protocol for DEC PDP-11 and VAX minicomputers. So XMODEM solves many problems; where does it fall short?

Mainframes

If you want to send files to a mainframe, or a minicomputer other than a DEC, then XMODEM will not help you. These machines just do not speak the correct protocol. Our solutions now are to change the protocol to one the mainframe understands, or implement one we know on the mainframe. Larry Hughes has a Fortran program which can be installed on a micro, mainframe, or mini to talk to his MITE package. He will send you a copy free if you ask for it when ordering MITE. This is a pretty good deal since it lets you talk to many machines for which no other simple solution is available. However, this gives you fairly slow communications – not very good for moving large amounts of data. Where are we to find a really good solution for micro-to-mainframe communications?

I mentioned earlier that many mainframes use protocols that were developed long before XMODEM was around. Some of these protocols show their age; XMODEM is better by far. Others were pretty well designed and offer capabilities and performance well beyond anything XMODEM is capable of. The mainframe protocols normally use a cyclic redundancy check (CRC) rather than a checksum; this gives much greater confidence in the accuracy of the received data.

Most of the common protocols allow blocks of data to be any number of bytes (or even any number of bits) long. This is more efficient than fixed block sizes, but introduces other problems. For example, because the block can be of any length, the protocol must either transmit the length in advance or mark the end of the block in some way. If the end of the

How Accurate Is Accurate?

by Leslie Brooks and John Rasp

When you are typing data or commands for the mainframe to process, you can tell immediately if you get a transmission error because the wrong character shows up on the screen. You then backspace over it and retype it. When large amounts of data are being sent between machines, this method of error checking is inefficient because of its character-by-character nature. Thus the protocols that have evolved usually provide some other means of detecting errors, such as checksums or cyclic redundancy checks. The next question is, "How good are these methods of error detection?" The answer can become very complicated; people have earned Ph.D.s in math for research in this area. I put this problem to a friend of mine who is doing doctoral work in statistics and he pronounced it "interesting." After a few days of study and research he told me that if we make a few assumptions, we can get a reasonably simple answer.

For the 8-bit checksum, there are actually three common ways of calculating it; they do not produce identical results. The first method is simply to add all of the data bytes together, modulo 256, and take the result as the checksum. The second method is the same, except that any carries generated are added back into the sum. The third method is to exclusive-or all of the data bytes together (sometimes called a longitudinal redundancy check). The best accuracy one can, in theory, hope for with an 8-bit checksum is to miss one error in 256. In practice, however, things may be less accurate, depending on the method used and the context in which the error occurs. Any simple calculation of the accuracy of these error-checking methods assumes that:

(1) There are about the same number of ones as zeroes in the data. The mathematics becomes much worse but the probability of detecting the error improves if this assumption is not true.

(2) The changes are independent of each other. This is not true – because phone line noise occurs in bursts, the fact that bit n was clobbered greatly increases the probability that bit n+1 will be clobbered also. My friend feels that this will not affect the result, particularly if assumption (1) is true. If assumption (1) is false *and* the changes are dependent, the math really gets hairy.

(3) There is an equal probability of changes in each direction. That is, saying that bit n was clobbered means the receiving system will see a one bit exactly half the time. This assumption should be true of noise, but may not be true of hardware faults. If it is false, the probability of detecting the error increases, but it is very difficult to say by how much.

After all of those caveats, if we assume that, on the average, ten bits get hit in any particular bad block, then the probability of missing an error is between one in 142 and one in 147. The longitudinal redundancy check misses approximately one error in 142. My friend was able to calculate this mathematically, but the math involved with the two addition methods was so complex that a Monte Carlo simulation was set up to estimate the accuracies. If the 8-bit checksum is performed by the simple addition of data bytes, modulo 256, approximately one error in 147 will be missed. Our confidence factor here was roughly 95 percent. When the carry is added back into the sum, the checksum is slightly less accurate.[1] This is acceptable for non-critical applications.

The cyclic redundancy check is a better error checking mechanism than the checksum. The mathematics are quite interesting; they are based on treating the message as a huge integer. The message is then (for a standard CRC-16) multiplied by 2^{16} and then divided by 11000000000000101 (the generator polynomial, 17 bits long). The 16-bit remainder is then attached to the end of a message as the error check code. This process is repeated on the receiving end to check for errors.[2]

The standard CRC-16 (sixteen-bit CRC) will detect all errors of 16 bits or less, and 99.955 percent of all errors of more than 16 bits.[3] This means that in the worst case (more than 16 bits in error) we will fail to detect the error only one time out of 2200. If we again assume that on the average only ten bits are changed, the probability of not detecting the error becomes even smaller.

References

[1] John Rasp, Statistics Department, Florida State University, Tallahassee, Florida 32306.

[2] John E. McNamara, *Technical Aspects of Data Communications*, Digital Press, pp. 145-158.

[3] McNamara.

block is marked, then this special end-of-block character must not appear in the data. This problem is usually solved by defining a transparent mode, in which the data is altered to remove any special characters before it is transmitted. For example, in HASP a DLE (Data Link Escape) character in the data would become DLE DLE, and the receiving station would throw away the first DLE. The common mainframe protocols are not simple!

These protocols usually require synchronous modems, and 2000 baud is the slowest speed commonly available – 4800 baud is very common, and 9600 baud is not unusual. If you really need to move a lot of data, this is the way to go. This approach is not cost-competitive with XMODEM or MITE, but it offers tremendous performance and easy entry to mainframe environments where the other packages are not available.

2780/3780

A typical mainframe protocol would be 2780/3780, an IBM protocol developed many years ago and now in widespread use by many other companies. It is available on virtually every mini and mainframe in existence. At 2000 baud, XMODEM's 128-byte blocks would take only half a second to transmit, so 2780/3780 allows for larger blocks. The larger blocks give much better line utilization, particularly at higher baud rates. However, 2780/3780 is not a simple protocol; it requires larger buffers than XMODEM (and uses much more memory overall). There are also plenty of places where the implementor can make mistakes. Still, it is a good protocol and very useful for moving large volumes of data around. It is primarily designed for remote batch stations and for moving data, not for interaction. The major disadvantage of this protocol is that only one thing can be happening at any given time. For example, the user can be printing a file from the mainframe, but cannot be downloading a file to disk at the same time.

HASP

HASP is another IBM protocol; it is very nearly a superset of 2780/3780. It is widely available on mainframes and minis of all types. HASP provides capabilities that are really incredible (the protocol defines seven printers, all of which can be going at once!). HASP also compresses data before it is sent, increasing the effective transmission speed. A good implementation of HASP will eat 48K of RAM in an instant, and one could easily use 64K or even more (if it allows large buffers, is fully menu driven, has on-line help, etc.). Most implementations on micros leave off a few things – like five or six of the seven printers. (When was the last time you had the need to run three printers simultaneously from your micro?)

HASP is also very complex, and there are many places to make mistakes. Again, a top quality implementation will anticipate and correct for mistakes in the *other* machine. I helped develop a package called HASTE for the computing center where I work and I know that the other implementations do make mistakes sometimes. In some cases (not all) clever programming on the micro end will correct faults on the mainframe end. For example, HASP compresses data before it sends it; in theory a single 400-character block (a common size) could expand into 16K of actual data! Needless to say, if you try to expand the received data in memory, you can get yourself into real trouble. HASTE expands the data only when it reaches its ultimate destination – the console, the printer, or disk. We have seen a mainframe implementation, however, which tries to do it in memory – we know because we blew it out of the water with a large compressed block!

HASP also defines control bits to control the different data streams (one stream for each of the seven printers, plus console, card reader, and card punch). These control bits may be transmitted with any block of data or as a separate message if there is no data to send. Thus when the micro's printer buffer fills up, we send out the FCS bits saying, "You can't send me any more printer data." Later, when the buffer empties, we send another message saying, "Printer stream one is now available again – send more data." If the micro receives a NAK (negative acknowledge) it must retransmit its last block.

What is not clear from the documentation is whether the retransmitted block should contain the previous FCS bits or the current FCS bits. If we send the previous FCS bits (saying "Send more data"), but our buffer is now full and the mainframe believes that FCS bits are always correct, then it will send data that we have no room for. If we send the current FCS bits, and the mainframe believes that FCS bits stay with the block (i.e., may not be current), then it will not act on FCS bits sent with a retransmitted block (they may be out of date). This means that we can send FCS bits saying, "Send me more data," and the mainframe will ignore them. We will then wait forever for the mainframe to send more data. The solution to this is to treat the FCS bits as always current, and retransmit them until they eventually go out on a block that is not itself being retransmitted. This will work with either implementation on the mainframe. Not every implementation (micro or mainframe) concerns itself with little details like this!

To give you a feel for the speed of a protocol (and package) like this, I recently downloaded 4.5 megabytes of data from our mainframe to the hard disk on a Kaypro 10. Using a 2000 baud modem it took just five hours.

HASP is again designed more for moving data than for interaction, but it allows concurrent operations. A single user can in theory be printing a file from the mainframe, sending the mainframe a file from disk, receiving some data and storing it on disk, and typing in some commands for the mainframe, all at the same time! Not all implementations will support this in the same way, but most provide at least some of these capabilities. HASP makes for a wonderful (although expensive) micro-to-micro file transfer environment. Because of the multi-stream capability you can type messages to your friend on the other end of the line at the same time that you are sending him or her the latest copy of the new utility you are working on. No more need for a second phone line or hanging up and redialing.

3270

As a last example of a popular protocol, 3270 is yet another IBM protocol that sees wide use. It is, however, not widely used outside of IBM sites. This protocol is designed for interactive work at a terminal; it thinks in screens. It is synchronous like HASP and 2780/3780, and can be either bit or byte oriented. HASP and 2780/3780 deal strictly with bytes – 3270 can transmit three bits as easily as eight. This protocol is not used much for moving data between systems, primarily because that is not what it does best. The 3270 protocol contains information on cursor positioning, screen clear, and other terminal-related things.

Conclusions

Communications is a very large, very deep swamp, and it contains many alligators. If you jump in with no preparation, you will get eaten alive. However, with a bit of thought toward what you wish to achieve, you can get through it successfully.

If cost is of utmost importance, look at XMODEM. If you have any sort of common micro, XMODEM is probably already available for it for free.

If you need to talk to a single mini or mainframe, and need to move moderate (less than 500K characters per week) amounts of data, take a good look at XCHANGE11, MITE and its Fortran friend, or some of the other similar packages which are available.

If you need to move large amounts of data, or need to talk to several different mainframes or minis, investigate the HASP and 2780/3780 synchronous packages. These are more expensive initially, but pay for themselves with their tremendous performance.

If you need to interact with an IBM mainframe, but don't need to move much data, look into 3270.

As with everything else, there are good packages available and also some very poor ones; investigate before you buy.

I have waved my hands over some of the details and completely ignored others, but I hope that overall this has given you a pretty good introduction to communications. If you now know the proper questions to ask and can tell when someone is trying to snow you (or just doesn't know the answer) then I will have accomplished everything I could reasonably hope for. I have no relationship to any of the companies or products mentioned, except for HASTE which I helped develop. DDJ

Reader Ballot
Vote for your favorite feature/article.
Circle Reader Service **No. 193.**

Unix-to-Unix Network Utilities

This article examines a relatively unknown yet powerful set of asynchronous Unix-to-Unix network utilities initially developed for internal use at Bell Labs and now available on most of today's Unix micro machines.

These network utilities provide perhaps the most standard (if not the simplest) method to move files and programs from one Unix machine to another. More importantly, a manageable and inexpensive network can be created to perform a variety of chores such as remote spooling, central filing and backup, remote data collection, electronic mail, and so on.

There are two basic utilities: uucp and uux. Uucp (Unix-to-Unix copy) is intended to work much like the Unix file copy utility, cp. The only difference is that the source and destination files also include the name of a Unix system on the network. The uucp syntax is:

```
uucp [-options] sys1!src sys2!dst
```

such that **src** and **dst** are source and destination filenames, and **sys1** and **sys2** are the designated machines on which the file exists or to which the file is copied. The entity in the square brackets is optional.

For example, the command

```
uucp A!file1B!file2
```

will copy **file1** on machine **A** to **file2** on machine **B**.

Uux (Unix-to-Unix execution) allows commands to be initiated on the local machine and executed remotely on another machine. A request to execute a command is sent to the remote machine, and if the request can be honored (i.e., the program exists on the remote Unix machine and is executable), the remote machine will try to execute the command.

The uux syntax is:

```
uucp A!file 1 B!file2


cmd2 | uux-sys1!cmd1 [sys2!files]
                    or
uux-sys1!cmd1[sys2!files]<[sys3!]file
```

In the first case, uux will execute the command **cmd** on the system **sys1** and, if necessary, will first copy the required **files** from system **sys2** to system **sys1** to be available to **cmd** on its command line. The next two cases are much the same except that uux's standard input is redirected to come from a Unix pipe (the second case) or a file (the third case); the rest of the arguments function the same as in the first case. Note again that the entities in the square brackets are optional.

The important thing to note about the last two cases is that uux's standard input is copied to the remote machine **sys2** along with the request to execute **cmd1**. Once on the remote machine, **cmd1** and its command line are reconstructed for proper execution.

The Unix-to-Unix network utilities include a number of other utilities that perform, among other things, the actual machine-to-machine communications (protocols, handshakes, log-in procedures) and the actual remote machine execution of the requested commands. Uucico (Unix-to-Unix copy-in copy-out) and uuxqt (Unix-to-Unix execute), respectively, perform these chores—typically in the background.

Each time a uucp or uux command is issued, a group of special "work" files is queued in a spool directory, and uucico is run in the background to process these files. The "work" files contain all kinds of information about which files are to be copied and to where, how the files should be copied, and how to execute the requested command. Since uucico is run in the background, the user does not need to wait until the actual work is completed before receiving control of the Unix shell again. As uucico works, it posts status checkpoints in a log file.

After finding a work request and checking permission for the request, uucico attempts to log on the remote machine. The entire log-in procedure is user-defined when the uucp system is first configured and can be tailored to perform almost any dial-out/log-in sequence.

If the Unix log-in prompt is successfully reached, uucico logs on the remote machine as a user named "uucp." Logging in as "uucp" causes the remote machine to execute a uucp command, which in turn forks a uucico command in "slave" mode; the uucico process that made the call to the remote machine runs as a "master." The slave process initiates the handshake and the master acknowledges. They agree upon a protocol and the master begins sending requests.

After the master has exhausted its requests, the uucico roles are switched—the master becomes the slave and the slave becomes the master. If the called machine has work bound for the calling machine, the new master sends its requests to the new slave. This continues until neither machine has any more work, at which point they hang up.

Since Lehman Brothers has several in-house Unix machines, we felt that with uucp and uux we could take these off-the-shelf utilities and build a network. Our first application was to create a printing network (using primarily uux) in which any user on any machine could access any printer on any machine.

Building a network was not as easy as it sounds because: (1) we wanted the existence of the uux utility to be completely transparent; (2) we wanted the print command interface to be consistent; and most difficult of all (3) we wanted the uux command to work unattended even in the face of such things as log-in failures, locked devices, and a few minor but potentially crippling uucico bugs that are easily handled manually.

The first and second goals were met by constructing a virtual printer. This is a channel through which the print data is piped to a uux process; uux then routes the data to the target machine.

Achieving the third goal took much more effort. Since uucico runs in batch fashion, the user cannot interactively assist uucico during the processing of the work files and log-in procedures. Some problems can be anticipated, but if the log-in fails, uucico simply records the failure in the log file and dies; the user receives no message and may never know if the request was fulfilled. Moreover, once the log-in fails, uucico locks all calls to the remote machine for about an hour. Another attempt to fulfill the request is not possible until uucico is restarted manually or until uucp or uux is issued again—after the hour of lockout time.

Worst yet, uucico sometimes dies for no apparent reason, even though the log-in was successful! When this happens, uucico fails to reset the permissions on the /dev/tty files and the spool directory (/usr/spool/uucp).

To resolve these problems, we first carefully chose a log-in sequence that has about a 7 to 10 percent failure rate. Next, we wrote a daemon that wakes up whenever files are being spooled remotely. This small program can reset the permissions on the /dev/tty files and spool directory

by Ron Coleman

Ron Coleman, Advanced Systems Group, Lehman Brothers, Kuhn, Loeb, Inc., 2 Broadway, New York, NY 10004, (212) 839-0965.

and knows how and when to remove the lock files and automatically restart uucico.

Figure 1 (below) illustrates this network. The user sees only Lp1, Lp2, Lp3, and Lp4. If a user sends a request to a local printer on his machine, the request is honored directly. Whenever a printer is specified that is not on the local machine, a special preprocessor shell script adds one line of routing data to the beginning of the input file, and the request is given to the virtual printer. The virtual printer reads the first line of the modified input stream and integrates that line in a uux command string. The actual print data is then piped to the uux process that uses the routing string to direct the print request to the remote machine. Although a remote print request has yet to fail, we estimate the average failure rate to be about 3 to 5 percent. DDJ

Reader Ballot
Vote for your favorite feature/article.
Circle Reader Service **No. 194.**

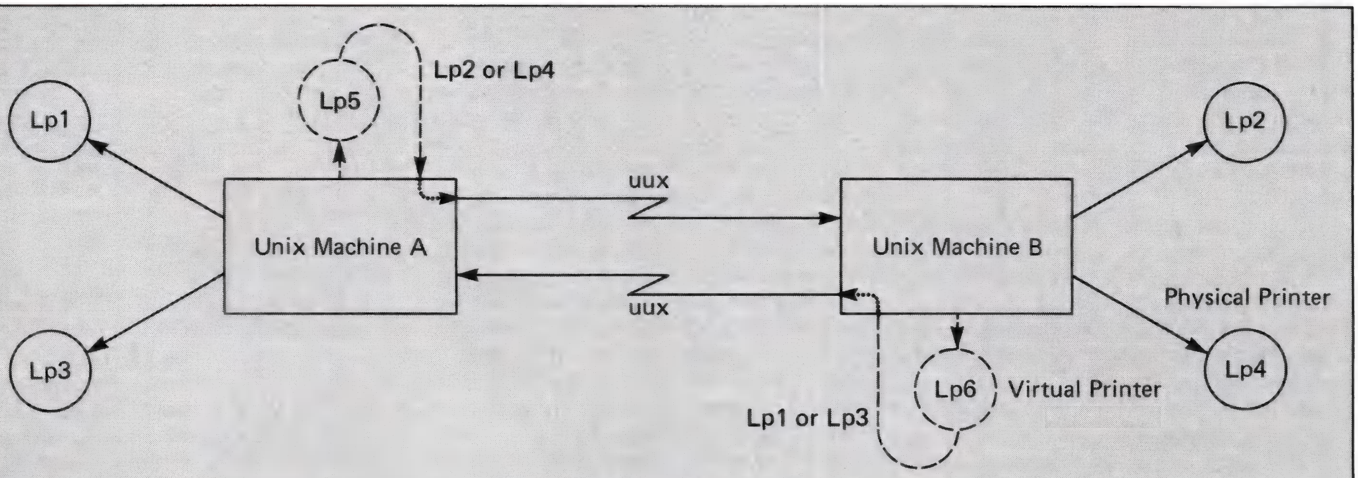


Figure 1.
Unix-to-Unix Network:
Physical/Virtual Printer Relationship

Virtual Personal Computer

The Virtual Personal Computer (VPC) project is an effort to develop a general architecture and specification for a personal computer that can be integrated with networks and larger computer systems. This article concentrates on the overall philosophies and architecture of the VPC.

The primary goal of the VPC project is to integrate personal computing and time-sharing. Time-sharing systems and personal computer systems have been based on fundamentally different philosophies. The Unix time-sharing system was designed to support a large number of dumb, full-duplex terminals. Typical Unix users are accustomed to the normal delays and slow communication links associated with such systems. Personal computers, on the other hand, have been designed as dedicated nontime-shared systems offering users unbelievable real-time response. Personal computer users that have never used a time-sharing system are being spoiled by this dedicated processing capability.

by Jim Fleming

Jim Fleming, Unir Corporation, 5987 E 71 St., No. 106, Indianapolis, Indiana 46220.

Software developed for time-sharing systems may be incompatible with personal computers and vice versa. As an example, in the Unix time-sharing system the concept of blocked I/O is used extensively. When a process wants to read information from a user's terminal, it issues a **read** system call. If the user has not typed any information, the process becomes **blocked**. The process will wait indefinitely until input is generated. While the process is blocked waiting for terminal input, it is not allowed to do any computation or read any other I/O devices. This blocking frees the main processor, which allows other users to share the same processor (thus, the term time-sharing).

Personal computer software is often designed without the concept of blocking. Instead, if input is desired from a user terminal, a check is done first to see if input is available. If input is available, a read is requested. The read system call returns immediately with the information without blocking because the check had indicated that data was available. If no data is available, another check can be made, and the user program will loop indefinitely waiting for inputs. If other I/O sources need to be serviced, the user program can check these multiple sources and read any that have data waiting. Keyboard input can be checked, a printer can be controlled, and a communication link can be serviced with precision under control of a user program.

As time-sharing and personal computer environments begin to converge, we must be aware that an indiscriminate merging of techniques from each environment may not yield an optimum situation.

The goal of the VPC project is to integrate the best of personal computing with the best of time-sharing. The VPC handles all local real-time-sensitive activities to give the user instant feedback. Programs running on the VPC are designed like classic personal computer programs with the check/read looping methods. Because it is assumed that a dedicated processor is available in the VPC, no concern is given to the processor cycles wasted using this methodology.

The application programs resident on the host system, however, are designed using the normal time-sharing blocked I/O techniques. Messages are exchanged between the VPC and the host at a rate compatible with these methods.

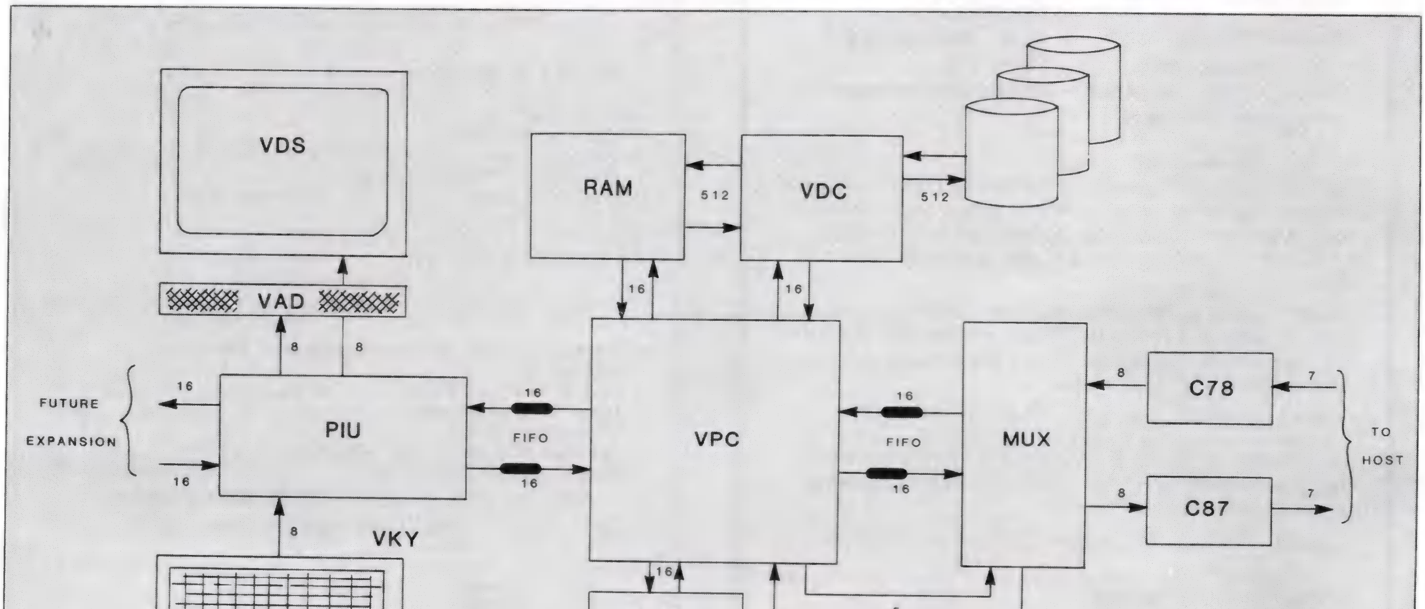


Figure 1.
Basic VPC Architecture

In systems like Unix, no change needs to be made to the application programs. The functions normally performed by the device drivers are "remoted" to the VPC. A simple change to the existing Unix device driver (which reduces its size) makes the VPC transparent to the application. The user is also unaware that functionality has been moved from the host into the VPC. High-speed real-time response is now possible so that personal computer users are not penalized by integrating their system with a time-sharing system.

Basic VPC Architecture

The basic VPC architecture is shown in Figure 1 (page 32). A process(or) is connected to the host system via a bi-directional 16-bit data path and a 5-bit control path. The 16-bit data path is called the remote I/O channel, and the 5-bit control path is called the remote control channel. The process(or) is interfaced to the user by another 16-bit bi-directional data path called the local I/O channel.

The process(or) is a 16-bit machine that executes one program residing in a 64K address space. The process(or) can be controlled by the host via the remote control channel. The process(or) can send and receive data to the host or user over the respective I/O channels. Local disk storage is supported using a hierarchical file-naming structure. An interval timer and time of day clock are also supported.

Character-oriented peripherals are connected to the local I/O channel. The upper eight bits of the local I/O channel are decoded as a device address, and the lower eight bits carry data to and from the device. Many protocols can be supported by these character-oriented peripherals.

The architecture of the VPC is based on the layered approach used in many data communication systems. Layering allows various parts of a system to be specified independently of the other parts. Changes can be made in a layered system with the confidence that the entire architecture will not fall apart.

Four distinct layers are evident in the VPC architecture. The four layers can be seen in Figure 1 by imagining that vertical lines separate the major boxes. Working from right to left, one can see the link layer, the multiplexer layer, the process(or) layer, and the user layer.

The Link Layer —
7-Bit, Asynchronous, Even Parity Version

The primary purpose of the link layer of a VPC is to guarantee that 8-bit bytes can be transferred in both directions in an error free (or error unlikely) manner. A variety of link layer configurations can be developed. Only the asynchronous parity-checked configuration will be discussed here because this is commonly used on personal computer systems.

In Figure 1, the link layer consists of the boxes C78 and C87 plus the hardware necessary to transmit and receive data to and from the host. In a 7-bit, even parity, asynchronous transmission system, a UART and modem are usually used as the primary hardware components. In this type of system, only seven data bits can be passed across the link. The software modules (C78 and C87) are used to convert one or more 7-bit data items into an 8-bit byte. The 8-bit bytes are used to interface the link layer with the multiplexer layer.

Many schemes have been devised to convert 8-bit bytes to other forms for transmission through 7-bit channels. The most common technique is to split each 8-bit byte into two 4-bit nibbles and send each nibble as an ASCII character from the set (0–9, A–F). This method works fine but doubles the amount of transmission. A 4000-byte file requires 8000 bytes of transmission.

Other methods block multiple 8-bit quantities together and enclose the packet between a header and a trailer. The data (the 8-bit quantities) are wrapped inside the packet and flow through the channel in a semi-transparent manner. Blocking does not help the problem of pushing 8-bit packets through 7-bit channels. Blocking can also be inefficient when many small data messages are sent because of the header and trailer overhead.

The filters shown in Figure 1 have been designed to push 8-bit bytes through 7-bit channels without the need for blocking and without interfering with control codes commonly used for flow control (e.g., XON, XOFF). The C87 filter (see Listing One, page 46) can be used to convert any 8-bit stream into a 7-bit stream. The C78 filter (see Listing Two, page 50) will convert the 7-bit stream back to 8-bit format. Each of the filters reads the standard input and writes on the standard output and can therefore be integrated with other Unix tools.

The filters operate by starting with the assumption that only 96 codes (32–127) are available for data. The lower 32 are assumed to be control codes, and the upper 128 are lost to the parity bit (normally even). The 96 codes are split into two parts. The codes 32–39 are used to encode one of eight base addresses. The eight base addresses are all offset by eight to account for the common occurrence of ASCII data in the channel. The offset of eight is assumed by both filters and not sent through the channel. The codes 40–127 are used to encode an offset from the base address.

The base address is changed only when

a code cannot be reached by the offset. If the base address has to be changed, one 7-bit data item is sent to change the base address, and one 7-bit data item is sent with the proper offset. If the base address does not have to be changed, then one 7-bit data item is sent with the offset. As long as the base address does not have to be changed, each 8-bit byte is encoded as one 7-bit data item. In normal practice one finds that ASCII text grows by about 25-30 percent and object modules grow by about 30-35 percent.

These filters are used to form the link layer processing in the VPC. The C87 filter processes incoming data, and the C78 filter processes the outgoing data. The normal Unix device drivers handle all flow control and do not interfere with data because the control codes are avoided. The 8-bit data emerging from the C78 filter are sent to the multiplexer and further decoded. When the multiplexer is ready to send an 8-bit value to the host, the C87 filter processes the byte. *All* data values (0-255) can be sent through this layer using these filters.

It should be noted that this link layer method is able to detect some errors in the transmission channel but is not able to correct any errors. This method can be used when a VPC is connected to a host via a hard-wired connection or a low-speed modem.

Future versions of the VPC will feature different link layer arrangements. Link layer methods for blocked asynchronous and synchronous transmission are being developed. These methods allow for error detection and correction via retransmission. They guarantee an almost 100 percent error free transmission. This change to the link layer will not impact other parts of the system.

Another item of note concerning the link layer is that, when a VPC is run resident on a single Unix system, no link layer processing is needed. The VPC multiplexer layer is connected via Unix pipes to the host multiplexer layer. Unix pipes are able to pass 8-bit bytes in an error free manner from one process to another.

As can be seen, a lot of options are possible in the link layer portion of the VPC architecture. Because of the layered design approach, changes in the link layer do not affect the other layers. In fact, in most sophisticated networked systems the link layer can be replaced by existing low-level transmission methods. If a system already supports X.25, ETHERNET, or another proprietary protocol, the link layer function of the VPC can be performed by the resident transport mechanism. The only function that the link layer performs is to provide an 8-bit error free path with flow control. If this can be provided in another manner, one is free to use that method. The other layers of the VPC are not affected.

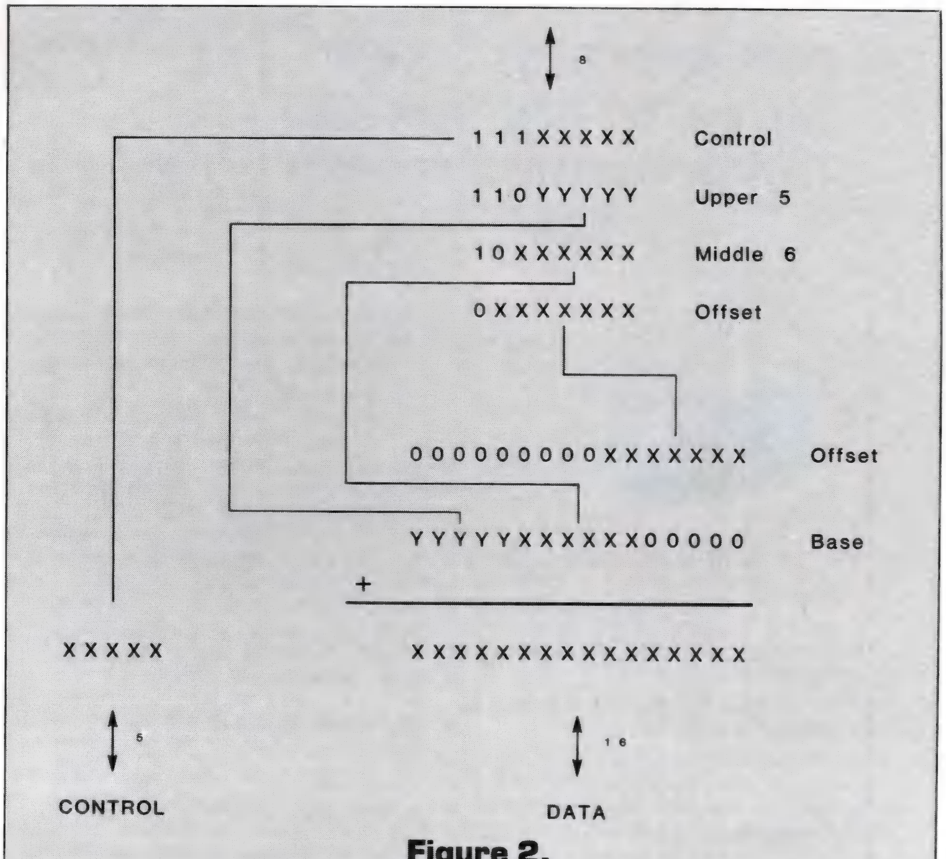


Figure 2.

Multiplexer Encoding Format

The Multiplexer Layer

The multiplexer layer is the heart of the VPC-to-host interface. The sole purpose of the multiplexer layer is to encode the 16-bit data from the remote FIFO and the 5-bit data from the remote FIFO (discussed at greater length in the next section) and the 5-bit data from the remote control channel into 8-bit bytes. These 8-bit bytes make up the interface to the link layer. The multiplexer encoding scheme is extremely simple and is shown in Figure 2 (below).

When a 0XXXXXXX (0-127) code appears on the 8-bit channel, the lower seven bits are added to a previously loaded 11-bit register, producing a 16-bit result (see Listings Three and Four, pages 38 and 39 respectively, and the discussion on Levels of VPC, page 44). The 11-bit register is padded with five zeroes (on the right) before the addition is performed. The 2-bit overlap between the two values means that the 7-bit code is actually a positive displacement from a base that is always a multiple of 32. The 7-bit code can actually reach 128 values from a given 11-bit base. The 11-bit base is changed only if this 128 position window is exceeded.

Initially 00000000000, the 11-bit base can be changed using 10XXXXXX or 110XXXXX. The middle 6 bits of the 11-bit base register are loaded when a 10XXXXXX (128-191) is received. The upper five bits of the base register are loaded when a 110XXXXX (192-223) is received. No data are produced when either type of code is received; the 11-bit base register is changed and saved for future use. All multiplexers should initialize the 11-bit base register to ensure that a known state is set when the link is established. The default value at start-up is 00000000000.

The codes 111XXXXX (224-255) are used for a 5-bit control channel. The 32 codes that form the control channel are divided into 16 opcodes (1110XXXX 224-239) and 16 data values (1111XXXX 240-255).

The end result of the multiplexer function is to create the illusion that a 16-bit bidirectional data path and 5-bit bidirectional control path connect the VPC and the host. If one desires to build this arrangement using ribbon cables and large connectors, the multiplexer can be removed. The other parts of the system are not impacted except that things may run a little faster.

The 16-bit data path is used for all data traffic between the VPC and the host. No subchannels are assumed, and any 16-bit value can be sent through the channel. Subchannels can be created by

Don't call her cheap. Call her beautiful.

The Bonnie Blue™

Word Processing System for the IBM Personal Computer

It's obvious what makes her so cheap, but what makes Bonnie Blue so beautiful? Bonnie Blue is a new and easy-to-use word processing program for the IBM Personal Computer.

The Full System. The Bonnie Blue System includes in one program a full screen Editor, a Printing module and a useful Toolbox. It includes the features you've come to expect, and more:

complete cursor control: by character, word, line; page up and down instantly; go to top, bottom of document; auto scroll towards top or bottom
word wrap
margin justification, centering
adjustable margins, tabs, indents
reformat paragraphs
move, copy, delete, paste blocks
find with delete, insert, replace and wild card characters
keyboard remapping
multi-line headers, footers

Bonnie Blue can handle lines longer than the screen is wide, by horizontally scrolling the line. And, unlike some programs, Bonnie Blue lets you include any displayable character in your text, such as block graphics and foreign language characters.

Unique Features. With Bonnie Blue, you can "paint" display attributes onto your text, by the character, word, or line, or automatically as you enter text. With the monochrome adapter, you can paint any combination of underlined, bold, reverse video or blinking. With an 80 column monitor and the color/graphics adapter, this translates into a palette of 16 color combinations to choose from. And if your computer has both monitors, Bonnie Blue lets you use them both, shifting back and forth as you wish.

Powerful Printing Module. You can use these colors or display attributes to highlight text on the screen, and Bonnie Blue can remove them from a file when you want (all files created by Bonnie Blue are DOS standard). The Printing module understands these text attributes, and you can map them into any single printer function or combination.

For example, normally you would want underlined text to print underlined. But you can tell Bonnie Blue to print underlined characters as both underlined and bold. Bright text on the screen can mean double struck, or emphasized and in italics. You are at the controls.

The first Print formatting module supports all the text capabilities of the Epson MX series with Graftrax Plus. By the time this ad appears, we will be supporting other popular dot-matrix and letter quality printers.

More than thirty "dot" commands give you added control of the format of your finished document. You can send it to a disk file instead of the printer, or preview the final page formatting on the screen.

Toolbox. The Toolbox is a set of useful functions, called "filters" that allow you to extract information from your files and transform their content. With these tools, you can join files together, sort lines of text, count words, find and substitute patterns, etc. Writers and programmers find this a useful collection of productivity enhancers.

Bonnie Blue is also great for a hard disk system. A thorough User's Guide, complemented by help screens and roadmaps, make the Bonnie Blue an exceptionally easy-to-learn and easy-to-use system.

Order yours today, or send for our free brochure. Bonnie Blue is available exclusively from **Bonnie Blue Software,** P.O. Box 536, Liverpool, NY 13088.

IBM Personal Computer is a trademark of IBM Corp. Epson Graftrax Plus is a trademark of Epson America Inc.

Only $50

Bonnie Blue Software

Post Office Box 536
Liverpool, NY 13088

□ Send me the Bonnie Blue System. I am enclosing $50 (NY State residents please add 7% sales tax).

□ Please send literature.
I have a ____________

□ Check enclosed □ VISA □ MasterCard Sorry, no COD.

Credit Card No.____________ Expires____________

Signature____________

Name____________

Address____________

City____________ State____________ Zip____________

Company____________

184

Minimum recommended system:
IBM PC, 128K, 2 disk drives, PC-DOS 1.1 or 2.0, 80-column monitor or monochrome adapter, or both, Epson MX-80 or MX-100 with Graftrax Plus.

Versions available soon for PCjr. Write for details.

Circle **no. 6** on reader service card.

conventions established for various applications.

A typical arrangement is to use the upper eight bits of the channel as a subchannel number and use the lower eight bits for data. This arrangement yields 256 independent 8-bit channels from the host to the VPC. The VPC may use a dozen of these channels for video windows, a couple for printers, and a couple for keyboards, and save the other 200 for future use. In noninteractive applications, 256 pipes can be established between the VPC and the host. This should be sufficient for most applications.

Another arrangement might be to use some of the high-order bits (five is a nice number) as a subchannel number and the other (three if five is used) high-order bits as subchannel data type indicators. For example, subchannel types for control and data could be determined by the type bits. With this arrangement, data arriving at the host can be classified as data or control and routed to the proper process or driver. Priorities can also be encoded in the upper bits so that channels can be handled with the proper urgency in the host. In all of these arrangements, eight free data bits are available in the low bits for total compatibility with existing systems.

The important thing to remember about the multiplexer is that its only job is to multiplex the 21 bits (16 data, 5 control) of information needed by the VPC process(or) into 8-bit bytes. No special assumptions are made about the 16 bits of data that flow through the multiplexer. Since the host has the ability to reconfigure the VPC process(or) via the 5-bit control channel, application-specific conventions can be established without impacting the multiplexer.

The VPC Process (or) Layer

The VPC process(or) layer interfaces to the multiplexer layer and the user layer via buffers called FIFOs (First In First Out). As the name implies, these 16-bit wide buffers are loaded and unloaded in the same order.

The remote FIFO interfaces the VPC process(or) to the multiplexer layer. Data written into the remote FIFO are eventually processed by the multiplexer and sent to the host. Data received from the host on the 16-bit multiplexer data channel are loaded into the remote FIFO and become available to the VPC process(or). The remote FIFO is bidirectional, and the data for each direction are kept independent of the other direction.

A similar arrangement exists for the local FIFO, which interfaces the VPC process(or) to the user layer. The user layer contains various device driver software modules and peripherals that interface with the user. The VPC process(or) can send and receive information to and from the user via the local FIFO.

The VPC process(or) is a classic 16-bit processor with 64K of data space. Program execution normally begins at location 0. The VPC process(or) contains a program counter, stack pointer, stack frame pointer, argument pointer, and literal pool pointer.

The VPC process(or) executes a portable instruction set tailored to the C programming language. System calls and interrupt handling are built into the instruction set and architecture of the VPC process(or). The system calls allow programs to access the local and remote FIFOs and the Virtual File System (VFS). Timing facilities are also built into the processor so that simple real-time applications can be implemented.

The VPC process(or) can be *completely* controlled via the 5-bit remote control channel. The host can start and stop the VPC process(or) and can read and write memory. In a typical situation the host downloads a small bootstrap program into the VPC process(or) memory space. The VPC process(or) is then told to execute this bootstrap program, which reads a larger bootstrap from the 16-bit data channel. The host sends the program over the 16-bit data channel. When the program has been sent to the VPC process(or), the host requests execution using the 5-bit remote control channel.

The host can also load a small program into the VPC that causes a file from the VFS to be loaded into the RAM and executed. This avoids the problem of lengthy download times from the host.

The 5-bit remote control channel can be used by the host to control the VPC process(or). In some respects the 5-bit control channel can be viewed as a remote front panel for the VPC process(or). The 5-bit values (sometimes called quibbles for 5-bit nibbles) are divided into control codes and data codes.

If the upper bit of the 5-bit value is a one, then the low four bits contain data. An internal 16-bit register (called the Temporary or T register) is loaded with the 4-bit values when they appear on the channel. The low four bits of a 16-bit word are sent first, followed by the next four, and so on. If more than four data values (16 bits) are sent, the extra data are dropped. This allows for upward compatibility with the 32-bit version of the VPC.

If the upper bit of the quibble is a zero, then the low four bits contain a control code. The control codes can be used to start and stop the VPC process(or), read and write the 64K of memory, and read and write the VPC process(or) registers. The 16 control codes are shown in Table I (page 37).

Two internal registers are used in conjunction with the remote control channel. The T register is a holding register for quibbles sent from the host; its operation was described above. The P (or Pointer) register is used as an auto-incrementing pointer to read and write memory. The T and P registers are *not* accessible by an application program running on the VPC process(or).

The typical sequence used to load data into memory or a register is to send the four quibbles of data followed by the opcode. The T register is zeroed *after* an opcode is executed, and any quibbles that follow are loaded starting in the low-order bits as previously described. To get everything in synch, one usually starts by sending a Write P Register opcode first. This gets the T register ready to accept quibbles. The data is then sent, followed by another Write P Register opcode, which loads the data just sent into the P register.

At this point the T register is again reset to zero, and data can be loaded using quibble codes. A Write Memory opcode is then sent, causing the contents of the T register to be written into the memory location(s) pointed to by the P register. The P register is auto-incremented by the number of bytes in the word of the VPC (currently two for 16-bit VPC). The contents of the T register are written into memory starting with the least significant byte.

When an opcode is sent that requests a read operation, the reverse action occurs. The contents of the desired source are sent, four bits at a time, followed by the opcode that was used to request the read. In other words, if a Read Program Counter opcode is sent, the contents of the program counter will be sent to the host followed by a Read Program Counter opcode.

When a Halt or Run opcode is received by the VPC process(or), the proper action is taken and the respective opcode is returned to the host as an acknowledgment of the request.

The VPC registers and memory can be read (and written) while the VPC is executing programs. The returned values represent a snapshot taken at the time the request reaches the VPC. If the registers or memory is changed while the VPC is running, unpredictable results can occur. The operations are well behaved, but no synchronization with the currently running program is guaranteed.

Programs that run in the VPC process(or) have complete control over the machine. Data can be exchanged with the host or peripherals without host intervention. Sophisticated, user friendly, front end processors can be developed, freeing the host from such time-consuming tasks. Because the VPC process(or) is totally dedicated to this single task, no delays are experienced by the user. The user has a dedicated personal computer connected to the host in an integrated manner. The host can utilize as much or as little of the power of the VPC process(or) as it desires.

The User Layer

The user layer is used to interface the VPC process(or) to the peripherals that are available to the user. A variety of peripheral combinations can be developed for various applications. The common peripherals are a video display, a keyboard, an audio device, a graphics display, and a serial printer. All of the peripherals can be controlled by the VPC process(or) through the local FIFO.

The local FIFO is similar to the remote FIFO. The VPC process(or) can read and write to the local FIFO at any time. Data written into the local FIFO is stored and eventually processed by the Peripheral Interface Unit (PIU). The PIU decodes the upper eight bits of the 16-bit word written into the local FIFO as a channel number. The lower eight bits are sent to the device (or device driver software) connected to the specified channel. By convention the lower bit of the upper eight bits is used to indicate control or data. Each device can be controlled independently of the data flowing between the device and the VPC process(or). When the bit is a zero a data path is indicated, and when the bit is a one a control bit is indicated.

This arrangement allows up to 128 devices (or drivers) to be connected to the PIU. Each device has a control and a data path. The control and data paths are both bidirectional. Communication between the peripherals and the PIU is completely asynchronous. No master-slave relationships exist except by local conventions established for specific applications.

The information exchanged is maintained in a FIFO queue. The queue can hold 256 16-bit words for each direction. Flow control is the responsibility of the VPC process(or) and the peripherals (or drivers). Peripherals should *not* allow the FIFO to fill as a means of indicating a "not ready" condition; the control channel should be used for this purpose. This prevents the system from completely locking up or blocking because a peripheral is not ready.

Three virtual peripherals or device drivers have been proposed initially for the VPC. The Virtual KeYboard (VKY) device is used for all key entry and supports a typical ASCII type of keyboard with function keys and special editing keys. The Virtual DiSplay (VDS) device is an 80 by 24 monochrome device with a simple window-oriented protocol. The Virtual AuDio (VAD) device is an 8-bit sound generator used to create audible tones and simple music.

The peripherals are connected to the PIU using channels 0 (VKY), 1 (VDS), and 2 (VAD). Note that the channel number is encoded in the upper seven bits of the local FIFO word. The low bit selects control or data as noted earlier. Therefore, the VKY device actually uses binary channel 0 (data) and 1 (control). The VDS uses 2 and 3, and VAD uses 4 and 5.

In the future more peripherals will be specified. Channel 3 has been reserved for the Virtual PRinter (VRP) driver, and channel 4 has been reserved for the Virtual GRaphics (VGR) driver. The VGR driver supports a subset of NAPLPS.

Even though any 8-bit character-oriented device can be connected to the local FIFO, a small number of compatible devices will be specified for anyone interested in working on a standard configuration. The following sections provide information on the three primary devices: VDS, VKY, and VAD.

0000	Halt the VPC Process(or)
0001	Run Starting at Location in PC
0010	Read Memory *P++
0011	Write Memory (*P++=T)
0100	Read P Register
0101	Write P Register (i.e., P=T)
0110	Read Program Counter
0111	Write Program Counter
1000	Read Stack Pointer
1001	Write Stack Pointer
1010	Read Frame Pointer
1011	Write Frame Pointer
1100	Read Argument Pointer
1101	Write Argument Pointer
1110	Read Literal Pointer
1111	Write Literal Pointer

Table I
Remote-Control Control Codes

0000xxxx	Control codes
0001xxxx	Window graphics
0xxxxxxx	ASCII characters
1xxxxxxx	Set buffer address

Table II
VDS Code Groups

VDS: The Virtual Display Device

The VDS device supports a simple window-oriented display compatible with 80 by 24 terminals. Partial screen scrolling is supported as well as business graphics for drawing simple forms.

The VDS protocol consists of 8-bit values sent to the VDS device in a stream format. The 8-bit bytes can be divided into the four groups shown in Table II (page 37). The VDS protocol can be implemented on most popular terminals and personal computers. Figure 3 (below) illustrates the control codes, window graphics, and ASCII characters supported by the VDS device.

VDS Control Codes

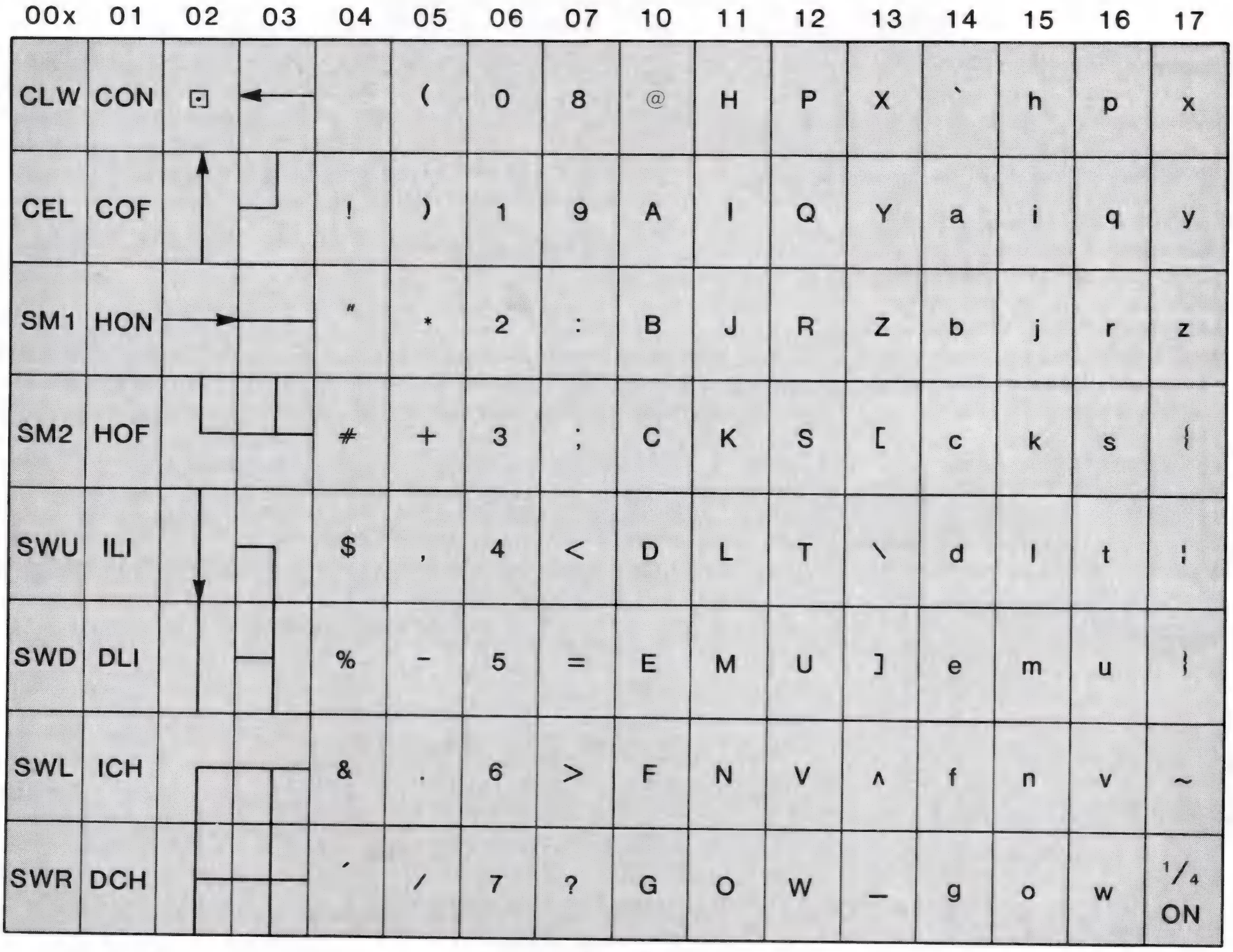
The 16 VDS control codes are listed in Table III (right). Most of the control codes are self explanatory. One unique feature is the concept of marks. A window is defined by two marks that determine a unique rectangular area on the screen. The marks are set by positioning the screen address to a specific place and

000	CLW	CLear Window
001	CEL	Clear to End of Line
002	SM1	Set Mark 1
003	SM2	Set Mark 2
004	SWU	Scroll Window Up
005	SWD	Scroll Window Down
006	SWL	Scroll Window Left
007	SWR	Scroll Window Right
010	CON	Cursor ON
011	COF	Cursor OFf
012	HON	Highlight ON
013	HOF	Highlight OFf
014	ILI	Insert LIne
015	DLI	Delete LIne
016	ICH	Insert CHaracter
017	DCH	Delete CHaracter

Table III
VDS Control Codes

0000	Centered dot
0001	Up arrow
0010	Right arrow
0011	Lower left corner
0100	Down arrow
0101	Vertical edge
0110	Upper left corner
0111	Left edge right tee
1000	Left arrow
1001	Lower right corner
1010	Horizontal edge
1011	Bottom tee
1100	Upper right corner
1101	Right edge left tee
1110	Top tee
1111	Crossing lines

Table IV
VDS Window Graphics Codes



	00x	01	02	03	04	05	06	07	10	11	12	13	14	15	16	17
0	CLW	CON	⊡	←		(	0	8	@	H	P	X	`	h	p	x
1	CEL	COF	↑	┘	!	)	1	9	A	I	Q	Y	a	i	q	y
2	SM1	HON	→	─	"	*	2	:	B	J	R	Z	b	j	r	z
3	SM2	HOF	└	┴	#	+	3	;	C	K	S	[	c	k	s	{
4	SWU	ILI	↓	┐	$	,	4	<	D	L	T	\	d	l	t	¦
5	SWD	DLI	│	┤	%	-	5	=	E	M	U	]	e	m	u	}
6	SWL	ICH	┌	┬	&	.	6	>	F	N	V	^	f	n	v	~
7	SWR	DCH	├	┼	'	/	7	?	G	O	W	_	g	o	w	1/4 ON

Figure 3.
VDS Codes

issuing a Set Mark 1 or Set Mark 2 code. Once the marks are set, the various window operations apply to the area bounded by the rectangle determined by the marks. In other words, to clear a portion of the screen, one sets up an area and issues a Clear Window code.

VDS Window Graphics

Window graphics are used to create forms, tables, windows, and so on. The 16 window graphics codes have the following basic format:

0001xxxx

The four low bits are used to independently set one of four segments.

Five of the window codes are not generally useful. The code 0000 would produce a blank character, which would be redundant with the ASCII space character (octal 040). The window graphics code 0000 should be displayed as a centered dot. This character can be used to indicate an unused space in word processing applications.

The codes 0001, 0010, 0100, and 1000 have only one segment lit and are not generally used in window and form drawing. These codes are displayed as arrows oriented so that the tip of the arrow exits the side of the cell that would normally have a lit segment. The code 0001 is displayed as an Up arrow.

The complete set of business graphics codes is listed in Table IV (page 38).

VDS Set Buffer Address

The buffer address (i.e., cursor position) is set using codes from 128 to 255. Direct and relative positioning is provided. Direct positioning is performed in relation to the physical display screen. Relative positioning is performed relative to the active window.

The codes 128–151 are used to set the relative address to the beginning of any of the 24 rows of the active window. Row 0 (i.e., code 128) is the first row of the window but not necessarily the physical screen. If a row is specified that is outside the active window, then a wraparound effect occurs. The column will be set to the left-most edge of the active window.

The codes 152–175 are used to set the buffer row address (0–23) independently of the current column and current window. The codes 176–255 are used to set the buffer column address (0–79) independently of the current row and current window.

VKY: The Virtual Keyboard Device

The virtual keyboard driver is an 8-bit device that produces 128 codes corresponding to the keys most commonly used in personal computer applications. As with other device drivers, the VKY has been designed to support very primitive keyboards by mapping common control keys to the VKY keys, yet it can support the exotic keyboards found on today's personal computers with a direct mapping of keys. A balance has been struck between the new and the old.

Figure 4 (below) illustrates the VKY (although this keyboard does not exist physically). Users should become familiar with the mapping of their physical keyboards to this functional set. A typical dumb terminal used with the Unix operating system usually maps the PAUSE to CTL-S, ENTER to CTL-D, and EXIT to CTL-C or DEL.

The VKY device has been designed to avoid having too many keys available for applications. Originally the intent was to develop a 256-key generic keyboard that would support everything but the kitchen sink. Several common keyboards were considered (i.e., IBM-PC, DEC-VT 100, DEC-PC, EPSON QX-10, etc.) from which a rather impressive set of keys can be assembled. But even though there are a considerable number of common keys (like A–Z), the lack of consistency was discouraging.

Several problems occur when the 256-key approach is used. The most serious problem is that the average user cannot remember 256 functional keys, much less the contortions that are involved to activate the keys. Another problem is that when 256 codes or functions are available, the application program must be more agile than when only 128 codes are used. Lastly, when 256 functional key-codes are specified, the mapping of those codes to the popular keyboards becomes a rather arbitrary and frustrating task. With such a large number of key-codes available, all judgment calls end up by supporting both keys even though they may be very close in function. For example, the keys PAGE UP (IBM-PC) and PREVious PAGE (DEC-PC) are obviously meant for the same purpose. When 256 codes are available, it is tempting to support both in case another computer arrives on the scene with both keys.

The alternate approach chosen for the VKY device was to limit the keyboard to 128 codes (seven bits). The problem with taking this approach is that standard ASCII codes fill the entire key table: the 95 visible character codes plus the 32 control codes plus DEL make up the 128 character set. It is obvious that the 95 visible characters should not be touched. The remaining 33 codes are more than adequate for function keys and editing keys, but the common ASCII control

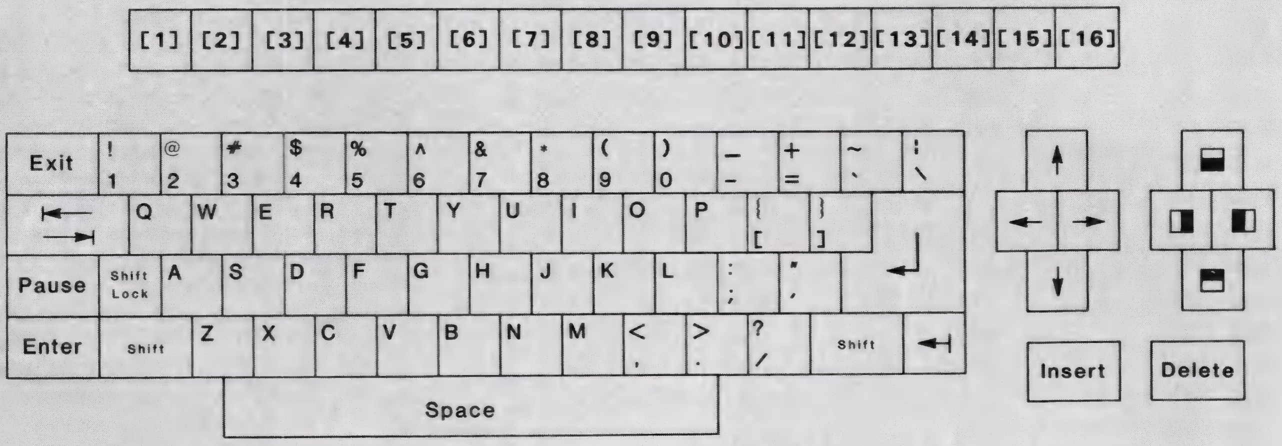


Figure 4.

VKY Functional Keyboard

000 - 003	Directional keys (arrows)
004 - 007	Page keys
010 - 017	Editing and control keys
020 - 037	Function/Menu keys ([1] to [16])
040 - 176	ASCII printable codes
177	Exit

Table V
VKY Code Groups

010	Backspace	(Destructive)
011	Tab	(Next field)
012	New line	(RETURN)
013	Back tab	(Previous field)
014	Pause	(Not automatic)
015	Enter	(Done, EOF, etc.)
016	Delete	(Character, Page, Window, etc.)
017	Insert	(see Delete)

Table VI
VKY Editing and Control Keys

000	Up	040	Space	100	@	140	`
001	Down	041	!	101	A	141	a
002	Left	042	"	102	B	142	b
003	Right	043	#	103	C	143	c
004	Begin	044	$	104	D	144	d
005	End	045	%	105	E	145	e
006	Previous	046	&	106	F	146	f
007	Next	047	'	107	G	147	g
010	Backspace	050	(	110	H	150	h
011	Tab	051	)	111	I	151	i
012	New line	052	*	112	J	152	j
013	Back tab	053	+	113	K	153	k
014	Pause	054	,	114	L	154	l
015	Enter	055	-	115	M	155	m
016	Delete	056	.	116	N	156	n
017	Insert	057	/	117	O	157	o
020	[1]	060	0	120	P	160	p
021	[2]	061	1	121	Q	161	q
022	[3]	062	2	122	R	162	r
023	[4]	063	3	123	S	163	s
024	[5]	064	4	124	T	164	t
025	[6]	065	5	125	U	165	u
026	[7]	066	6	126	V	166	v
027	[8]	067	7	127	W	167	w
030	[9]	070	8	130	X	170	x
031	[10]	071	9	131	Y	171	y
032	[11]	072	:	132	Z	172	z
033	[12]	073	;	133	[	173	{
034	[13]	074	<	134	\	174	|
035	[14]	075	=	135	]	175	}
036	[15]	076	>	136	^	176	~
037	[16]	077	?	137	_	177	Exit

Table VII
VKY Key-codes

codes have to be eliminated. This is not a major penalty because many users are confused when confronted with cryptic CTL-xx input techniques.

In fact, it turns out that many of the existing ASCII control codes can be preserved because of their widespread support on keyboards. New codes can be dropped in among the preserved codes, resulting in a very terse set of usable keys that almost any major personal computer can support in a manner that is obvious to a novice user. A side benefit of this approach is that those systems that have a primitive keyboard can use the CTL-xx combinations to generate the VKY codes because CTL-xx codes are not normally supported.

Another benefit of 7-bit (128) coding is that the high bit is now free for use as a KEY-SPEED bit. The VKY device sets the high order bit to zero when keys are struck in a normal manner. The high bit should be set to a one when a key is generated less than one tenth of a second after the previous key. The value in the 16-bit VTM timer is saved each time a key is generated. If another key is generated and the value in the VTM timer has not changed, the high bit of the new key is set to one.

The KEY-SPEED bit is normally set when a user is striking a single key at a rapid rate or when a special device (like a joystick) is being used to generate key strokes. Application programs can always ignore the high bit, the only possible penalty being read-time response. The presence of the high bit, however, can be used by the application program to indicate that it would be wise to raise the priority of keyboard processing since keys are being pressed at an unusually rapid rate. The functionality of each key should *not* be changed by the presence of the KEY-SPEED bit.

Keyboards and terminals with auto-repeat functions generate multiple keys when keys are continuously held. Keys are usually repeated 15 to 30 times per second. The high-order bit periodically will be set to one when auto-repeating keyboards are used. This is normal, and the application program can use this information to the best of its ability.

If special devices (like joysticks or touch pads) are used to generate multiple keys, then the KEY-SPEED bit should be set to indicate that a high-speed key input device is being used. Again, the KEY-SPEED bit should be used to indicate not a change in functionality but rather a change in key generation speed.

Various studies and large amounts of head holding have resulted in the following specification for the VKY device driver. One must keep in mind that the spirit of this effort is to identify functionality and not key-top identification. The VKY device keys can be grouped into the six groups shown in Table V (above).

USE MAGIC/L™ TO WRITE JUST ONE PROGRAM

Bet you can't stop with one.

For just $295 MAGIC/L™ is yours. And suddenly you find you've bought more than a language. MAGIC/L is an entire interactive environment; an assembler, a compiler, an editor, an I/O package, and a system call facility are all wrapped up and delivered. And now MAGIC/L is available for CP/M-based computers.

MAGIC/L is easy to learn. It has syntax similar to C and Pascal, and because it's extensible as well as interactive, it dramatically increases productivity.

Program development features include a built-in text editor, command line recall, CCP, STAT, and PIP command emulation, and the ability to store keyboard dialog on disks.

Key language features include: CHAR, INTEGER, LONG, REAL, and String data types; record structures similar to the STRUCT facility in C; and a complete I/O package that can provide random access, variable length I/O to any CP/M file.

And MAGIC/L offers great portability. Source code which runs on CP/M can be compiled unmodified and run on any other processor.

Typical applications include hardware interfacing, process control, games creation, interactive graphics and image processing. MAGIC/L has made programming easier for DEC, 68000, and Data General users. Now it's working for CP/M users too.

MAGIC/L provides everything you need to write a complete program. But the only way to be convinced is to try it yourself. Send us your $295 check or money order—we also accept MasterCard and VISA—we'll send MAGIC/L for CP/M to you at once. A full money back guarantee is part of the package. Once you've sampled that first program, you'll have to try another . . . and another . . . and another. MAGIC/L . . . it's more than a language.

MAGIC/L . . . It's more than a language

LOKI ENGINEERING, INC.
55 Wheeler St., Cambridge, MA 02138 (617) 576-0666

Circle **no. 40** on reader service card.

MAGIC/L is a trademark of Loki Engineering, Inc.
DEC is a trademark of Digital Equipment Corporation.
CP/M is a trademark of Digital Research, Inc.

VKY Directional Keys

The directional keys are the first four key-codes. The four directions (up, down, left, and right) are provided. Arrow keys are usually used to generate these codes. When diagonal movements are desired, a two-code sequence should be used with the KEY-SPEED bit set in the second key to indicate the composite action: if the application program makes use of the KEY-SPEED bit, it can interpret the two moves as a single move.

Users with only four arrow keys can achieve the same result by pushing two of the direction keys in rapid succession. Sophisticated keyboard detection schemes can also be used to allow a user to push two arrow keys at the same time. The resulting codes sent to the application program are the same.

If joysticks, mice, or touch discs are used, an appropriate stream of direction keys should be generated.

VKY Editing and Control Keys

This group of eight keys is probably the most controversial. This group has been formed as a result of careful selection with an eye toward the past, present, and future. The functionality of the keys may not be as controversial as the recommended physical keyboard equivalents. The functional codes are shown in Table VI (page 40).

VKY Function Keys ([1] to [16])

The codes 020–037 are very straightforward. They are used to encode special function keys that are used by an application in a specific manner. It is expected that these codes should be generated by physical keys that can be relabeled for the application. In units that have only 10 function keys (like the IBM-PC), it is anticipated that some CTL or ALT arrangement will be used to generate all 16 keys. When terminals like the VT-100 are used, the typical approach is to convert the numeric keypad into a function key cluster. When this approach is used, a two-key sequence can be used for each function key. The function keys [1] to [9] are actually generated by the sequences [0][1] to [0][9]; the function keys [10] to [16] can be generated by two-key sequences [1][0] to [1][6]. When the [0] key is pressed, the device waits for another key to make a final determination.

Another approach that can be used with a numeric keypad is to map the keys [1] to [9] to the function keys [1] to [9], with the [0] key treated as function [10]. The keys [11] to [16] are generated by the two-key sequences [1][1] to [1][6]. This approach requires that timing detect the difference between [1] and [1][X] combinations. This can be accomplished by starting a timer whenever the [1] key is pressed. If another key does not follow in a short period of time (maybe one second), the [1] function key is generated.

The latter approach to using the numeric keypad is usually more popular because the majority of the function keys can be generated by a single key-stroke. Other methods can be devised; it does not matter how the function keys are generated as long as the proper key-codes are generated for the application program.

VKY ASCII Key-codes

The codes 040–176 are equally straightforward. They are generated by the common ASCII punctuation and alphanumeric keys. The physical Shift key is typically used to access the upper-case and lower-case characters.

It should be noted that keys like Shift, Shift Lock, and Numeric Lock are all considered to be private to the keyboard and/or VKY device driver. The application program does not need to know precisely how the key was generated: it is concerned only with the intent of the user.

VKY Exit Key

The last bit of functionality needed in most applications is the Interrupt or Attention capability. The key-code 0177 is used to encode this function. The key has been named Exit to balance out the Enter key and some of the terminology from the system.

The physical DEL (Delete) key can be used for the Exit function although the ESC (Escape) key can be equally useful. Note that the normal ASCII ESC code 033 is used by the F12 special function key, which means the physical ESC key is now available. The CTL-C combination is another popular key used to generate the Exit key-code.

If the distinction between functional (logical) and physical keys is still confusing, the following example may help. If the ESC key is chosen for the Exit function, then each time the ESC key is pressed the VKY device should send the VPC processor an 0177 code indicating an Exit. The application program running in the VPC should *always* interpret the 0177 key-code as an Exit request whether or not it knows that the ESC key was used to generate the code. Alternate physical keys that can be used for Exit are the BREAK key or the DEL key. The DEL key should be used as a last resort because it is more commonly used to generate the Delete key-code (octal 016).

Each of the keys shown in Figure 4 generates a key-code. Table VII (p. 40). illustrates all of the key-codes supported by the VKY device. The key-codes should be handled by an application program as functional codes rather than as physical codes. The physical keys that are pressed do not have to be precisely labeled with the key-codes in the table. The user should be concerned with the desired function rather than the physical keys. This emphasis on functional keys allows application programs to be portable across a variety of machines.

	00		01		10		11	
0000	A	55.00	C#	138.59	F	349.23	A	880.00
0001	A#	58.27	D	146.83	F#	369.99	A#	932.32
0010	B	61.73	D#	155.56	G	392.00	B	987.76
0011	C	65.41	E	164.81	G#	415.00	C	1046.52
0100	C#	69.29	F	174.61	A	440.00	C#	1108.72
0101	D	73.42	F#	184.99	A#	466.16	D	1174.64
0110	D#	77.78	G	196.00	B	493.88	D#	1244.52
0111	E	82.41	G#	207.50	C	523.25	E	1318.52
1000	F	87.31	A	220.00	C#	554.36	F	1396.92
1001	F#	92.50	A#	233.08	D	587.33	F#	1479.96
1010	G	97.99	B	246.94	D#	622.26	G	1568.00
1011	G#	103.75	C	261.63	E	659.26	G#	1660.00
1100	A	110.00	C#	277.18	F	698.46	A	1760.00
1101	A#	116.54	D	293.66	F#	739.98	A#	1864.64
1110	B	123.47	D#	311.13	G	784.00	B	1975.52
1111	C	130.81	E	329.63	G#	830.00	C	2093.04

Table VIII
VAD Note Codes

VAD: The Virtual Audio Device

The Virtual Audio Device is an 8-bit device that allows a VPC program to generate simple sounds and music. Each 8-bit value written to the VAD device is used to immediately start a note of a particular frequency. The low six bits of the 8-bit byte are used to specify one of 64 notes. The upper two bits indicate the duration of the note.

The 64 notes correspond to the most common notes found in music. The notes range from A (note 0) two octaves below middle C to C (note 63) three octaves above middle C. Only the legal piano notes are supported. This arrangement allows six bits to represent tones from 55.0 Hz to 2953.0 Hz. The 64 notes are shown in Table VIII 9 (page 42) along with the frequency in Hz (cycles per second). The labels on the top of each row represent the high two bits of the note code.

The upper two bits of the 8-bit VAD code are used to control the duration of the tone. Table IX (at right) illustrates the four possible duration settings. The timing of the VAD tones is synchronized to the VTM timer. The VTM timer is a free-running 16-bit timer that is used by the VPC process(or) to schedule events and for other timing applications. The VTM timer ticks (i.e., increments) every one tenth of a second.

When an 8-bit byte is written to the VAD sound generator, the sound is instantly started. The note is played until one, two, three, or four ticks of the VTM timer occur. If a previous sound is still being played when a new value is loaded, the old sound stops and the new sound begins.

Note that if a sound is started very close to the next tick of the VTM timer and only one tick is requested then a short note may be heard. Normally, the VAD device is loaded as part of the VTM interrupt handler, which is executed at the beginning of a VTM time interval. This more or less guarantees that a full tenth of a second note will be generated. When two, three, or four ticks are requested, the notes will play for approximately two, three, or four tenths of a second, respectively.

The automatic turn-off feature of the VAD sound generator is extremely useful when a host computer wishes to send an isolated note to a terminal to alert the user. The host program can address the code to the VAD data channel, and the sound will be loaded, played, and ended without further codes being sent from the host.

00	Turn off on next tick
01	Turn off on tick after next
10	Turn off on third tick
11	Turn off on fourth tick

Table IX
VAD Duration Codes

The VAD device can be used to generate the common ASCII BEL code for backward compatibility with existing applications. The code for middle C (00011011) is normally used.

On systems without a general purpose sound generator, middle C is usually detected and converted to the BEL character on the terminal being used. When music is sent to a VPC, the occasional occurrence of middle C in a song will ring the BEL. The user may not be able to enjoy the song, but at least something is heard to indicate music is being played. The user is usually given the option to turn this feature off in the Set-up capability local to a VPC.

The Levels of VPC

At the present time three basic VPC configurations are defined. For a lack of better names, they have been designated level 0, level 1, and level 2.

The level 0 VPC is a terminal-only implementation. The local and remote FIFOs are patched together so that the host can communicate with all of the serial device drivers. The 5-bit control channel is not connected to anything and will not respond. No downloading of software can be performed. No files can be transferred.

The level 0 VPC can be used in applications that require a simple audio, video, and keyboard combination. Up to 128 serial devices can be handled using the standard FIFO driver interface techniques. Each key that is generated by the VKY is sent to the host. Likewise, all VDS screen control must come from the host.

A level 0 implementation of VPC has been developed by Unir Corporation for members of The Unir Project. It is available to members for $50.00 and includes a complete C language source code listing. Even though the package has been written for the industry standard VT-100 terminal, it can be easily adapted to other terminals.

Figure 5 (below) shows the modules included with the level 0 VPC package. The listings included with this article comprise the two rightmost elements (C78 and C87) and the two parts of the level 0 VPC (MUXPIU and PIUMUX). The relatively lengthy code for the VDS and VKY is not included. *[If readers show sufficient interest, we will run the code for those modules in a future issue. – Ed.]*

The level 1 VPC is a diskless version with all of the level 0 features plus the 64K RAM and VPC processor. Downloading can be done from the host using the 5-bit control channel in conjunction with the remote FIFO.

The level 1 VPC boots in the level 0 terminal mode. The local and remote FIFOs are patched together, and the VPC processor begins looping and checking the two FIFOs. Data that appear on one FIFO are transferred into the other.

The level 2 VPC is equipped with a file system (VFS) for local file storage. Using the VPC processor, files can be stored and retrieved. A standard Unix-like hierarchical file system is assumed by the system calls.

The level 2 VPC boots from a file designated during the local setup phase of VPC. This file is loaded into the RAM (using the exec system call) at location 0 and given control. The standard default boot file brings the VPC up running the same program as level 0 and level 1.

The VPC level can be detected in a variety of manners. The first step is to attempt to read the program counter using the 5-bit control channel. If no response occurs, a level 0 VPC is being used. If a response comes back, a level 1 level 2 VPC is in use.

Further memory interrogation can be done to determine if a level 2 VPC is being used. As a convention, the level 2 VPC will have argc and argv set during the file loading process. The name of the file can be determined by interrogating the argument stack. In contrast, the level 1 VPC will not have the file name on the stack.

Testing Your VPC

Unir Corporation is providing a free data line for people interested in testing their VPC implementations. The data line features a member directory, a bulletin board service, a news service, and several VPC exercise programs.

A level 0 VPC is needed to access the system. A 7-bit asynchronous, even parity data link is used to simplify the access. The system can be reached 24 hours a day by dialing (317) 842-7014.

More information on VPC, the VPC data line, and The Unir Project can be obtained from Unir Corporation, 5987 E. 71st Street, Suite 106, Indianapolis, Indiana 46220.

Conclusion

The virtual personal computer is a general system for integrating timesharing and personal computer systems. The VPC replaces the common CRT and keyboard with a complete programmable workstation. The primary source of programming for the VPC comes from a host system and not the user.

The VPC can be used as a simple terminal or as a powerful component in a distributed processing system. Programs can be downloaded from a host into the VPC memory and can be executed as high-speed, real-time tasks. Files can be transferred between the VPC and the host without extra hardware or software. A wide variety of peripherals can be controlled by the VPC using a general purpose I/O capability. The user interfaces to the VPC and ultimately to the host, through these peripherals.

Unlike many software systems, a VPC makes no prior assumption about how these various capabilities are going to interact. Because of this, a VPC can be configured as a multi-screen terminal emulator, as a real-time front-end processor, or a simple personal computer. Concepts like windows and concurrent, multiplexed I/O are easily integrated into a system using a VPC. As the system evolves, the VPC can be reconfigured to meet the demands of new applications.

The VPC is extremely useful in bridging the gap between personal computers and large, time-shared systems. This gap represents one of the next major frontiers in the computer industry. ■■J

(Listing begins on page 46)

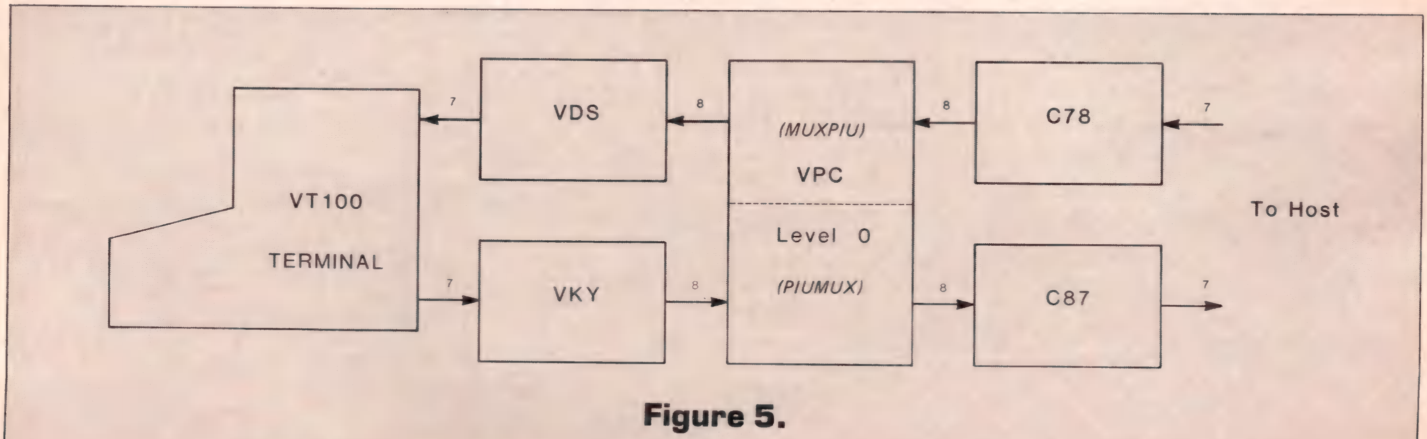


Figure 5.
Level 0 VPC

SWIG©

SOFTWARE WRITERS INTERNATIONAL GUILD

SCHEDULED SWIG ACTIVITIES & MEMBERSHIP BENEFITS

(1) $10,000 PROGRAMMING CONTEST (Members only)

(2) NATIONAL COMPUTER WEEK (May 11-May 20, 1984)

(3) ANNUAL CONFERENCE AND SOFTWARE AWARDS CEREMONY (During National Computer Week)

(4) CONSULTANT REGISTRY (With computer store referral system for customized software)

(5) JOB PLACEMENT SERVICE (Free to individual members, fixed maximum fee to companies)

(6) FREE SEMINARS & MEETINGS LOCALLY

(7) SOFTWARE LIBRARY LENDING & EXCHANGE SERVICE (Professional quality assemblers, utilities, games, etc.)

(8) SOFTWARE LOCATION SERVICE (For companies & individuals-if it exists, **SWIG** will find it. If not, see #9)

(9) SOFTWARE DEVELOPMENT SERVICE (From novice to scientist, **SWIG** members can work on any project-from applications to games to R&D)

(10) LEGAL SERVICE

(11) AGENT (**SWIG** can represent you in sales to software publishers)

(12) 24 HOUR - 7 DAY BULLETIN BOARD SYSTEM (BBS) ACCESSIBLE BY COMPUTER FREE

(13) AND MORE!!!!

THE LARGEST PAID MEMBERSHIP PROGRAMMERS GUILD - OVER 5,000 MEMBERS WORLDWIDE!!

MEMBERSHIP APPLICATION FOR **SOFTWARE WRITERS INTERNATIONAL GUILD**

NAME ______________________________

ADDRESS ______________________________

CITY ______________ STATE __________ ZIP __________

PHONE # (   ) ______________

- CLASSIFICATION:

□ NOVICE □ BEGINNER TO ADVANCED

□ ADVANCED WITH ON THE JOB EXPERIENCE □ RESEARCH/SCIENTIST

- WHAT EQUIPMENT DO YOU HAVE EXPERIENCE WITH &/OR ACCESS TO &/OR PLAN TO BUY?

□ MAINFRAME □ MINI □ MICRO □ DESIGN/R&D

BRAND NAME(S): □ IBM □ XEROX □ APPLE □ TI

□ COMMODORE □ RADIO SHACK □ ATARI □ OSBORNE

□ TIMEX/SINCLAIR □ NORTH STAR □ HEWLETT PACKARD

□ OTHER ______________________________

- AREAS OF INTEREST:

□ DATA PROCESSING □ BUSINESS APPLICATIONS □ GRAPHICS

□ LEGAL □ VOICE □ MEDICAL □ APPLIANCE (HOME) CONTROL

□ ROBOTICS □ GAMES □ MUSIC □ R&D □ OTHER ______

- MEMBERSHIP ACTIVITIES AND SERVICES OF INTEREST:

READ THE LIST ON THE LEFT AND CIRCLE THE NUMBERS BELOW THAT APPLY.

1 2 3 4 5 6 7 8 9 10 11 12

□ I HAVE ENCLOSED $20 ANNUAL MEMBERSHIP FEE □ CK □ MO
(MAKE CHECK PAYABLE TO: **SWIG**)

RETURN TO: **SWIG**
P.O. BOX 87
STONY POINT, NEW YORK 10980
(914) 354-5585

SWIG© SOFTWARE WRITERS INTERNATIONAL GUILD

Circle **no. 76** on reader service card.

Virtual Personal Computer (Text begins on page 32)

Listing One

```
#include <stdio.h>

/*
*
*       convert 8 bit mux format (mux)
*       to 7 bit even parity format (c7e)
*
*/

/*
*
*       even parity table
*
*/

char parity[] = {
        0x00,0x80,0x80,0x00,0x80,0x00,0x00,0x80,
        0x80,0x00,0x00,0x80,0x00,0x80,0x80,0x00,
        0x80,0x00,0x00,0x80,0x00,0x80,0x80,0x00,
        0x00,0x80,0x80,0x00,0x80,0x00,0x00,0x80,
        0x80,0x00,0x00,0x80,0x00,0x80,0x80,0x00,
        0x00,0x80,0x80,0x00,0x80,0x00,0x00,0x80,
        0x00,0x80,0x80,0x00,0x80,0x00,0x00,0x80,
        0x80,0x00,0x00,0x80,0x00,0x80,0x80,0x00,
        0x80,0x00,0x00,0x80,0x00,0x80,0x80,0x00,
        0x00,0x80,0x80,0x00,0x80,0x00,0x00,0x80,
        0x00,0x80,0x80,0x00,0x80,0x00,0x00,0x80,
        0x80,0x00,0x00,0x80,0x00,0x80,0x80,0x00,
        0x00,0x80,0x80,0x00,0x80,0x00,0x00,0x80,
        0x80,0x00,0x00,0x80,0x00,0x80,0x80,0x00,
        0x80,0x00,0x00,0x80,0x00,0x80,0x80,0x00,
        0x00,0x80,0x80,0x00,0x80,0x00,0x00,0x80
};

int lbase = 0;          /*      last base                       *//


/*
*
*       8 bit characters are read that contain
*       8 bits of data
*       they are converted to 7 bit format and
*       even parity is set in the upper bit
*
*/


main(argc,argv,envp)
int argc;
char *argv[];
char *envp[];
{
        int c;


        setbuf(stdin,NULL);
        setbuf(stdout,NULL);
        new win(0);
        whiTe((c=getchar()) != EOF){
                put_c7e(c);
        }
}

/*
*
*       set up new window
*
*       the offset of 8 is not encoded
*       but is set for later use
*
*/


new_win(byte)
int byte;
{
        lbase = ((byte-8) & 0xe0) + 8;
        put_par((lbase>>5) + 32);
}

/*
*
*       convert 8 bit bytes
*       to 7 bit format
*
*       7 bit format
*
*       3 bit base + 8       XXX01000
*       6.45 bit offset    + 0xXXXXXX
*                          -----------
*       8 bit result         XXXXXXXX
*
*       the 3 bit base is encoded
*       as numbers in the range
*       32 to 39
*
*       the extra offset of 8 is added
*       to provide more efficiency
*       when handling ASCII files
*
*       the extra offset of 8 is not
*       encoded but is assumed as
*       shown above
*
*       the 6.45 bit offset is encoded as
*       numbers in the range 40 to 127
*
*       the 3 bit base is only changed
*       when necessary
*
*       if consecutive 8 bit numbers
*       fall in the same range then the
*       3 bit base is "shared" and
*       each 8 bit byte will only require
*       1 byte containing a 6.45 bit offset
*
*       the codes 0 to 31 are never
        enountered
*       so that control codes can be used
*       for flow control and other purposes
*
```

(Continued on next page)

Virtual Personal Computer

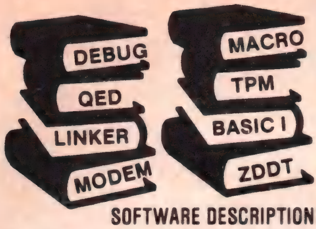
Listing One

(Listing continued, text begins on page 32)

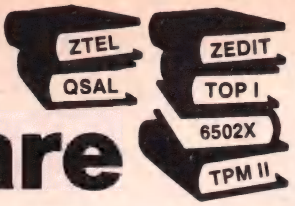
```
*       the codes 128 to 255 are not used
*       the high order bit is available for
*       a parity bit if needed
*
*/

put_c7e(byte)
int byte;
{
        if(((byte-lbase)&0xff) > 87){
                new_win(byte);
        }
        put_off(byte-lbase);
}

/*
*
*       output 6.45 bit offset
*
*       the offset should be in the range
*       0 to 87
*
*/

put_off(offset)
int offset;
{
        put_par(offset + 40);
}

/*
*
*       output 7 bits with even parity
*
*/

put_par(c)
int c;
{
        c &= 0x7f;
        putchar(c | parity[c]);
}
```

End Listing One

(Listing Two begins on page 50)



Z80 Software



SOFTWARE DESCRIPTIONS

TPM (TPM I) - $80 A Z80 only operating system which is capable of running CP/M programs. Includes many features not found in CP/M such as independent disk directory partitioning for up to 255 user partitions, space, time and version commands, date and time, create FCB, chain program, direct disk I/O, abbreviated commands and more! Available for North Star (either single or double density), TRS—80 Model I (offset 4200H) or II, Versafloppy I, or Tarbell I.

TPM-II - $125 An expanded version of TPM which is fully CP/M 2.2 compatible but still retains the extra features our customers have come to depend on. This version is super FAST. Extended density capability allows over 600K per side on an 8" disk. Available preconfigured for Versafloppy II (8" or 5"), Epson QX-10, Osborne II or TRS-80 Model II.

CONFIGURATOR I

This package provides all the necessary programs for customizing TPM for a floppy controller which we do not support. We suggest ordering this on single density (8SD).

Includes: TPM-II ($125), Sample PIOS (BIOS) SOURCE ($FREE), MACRO II ($100), LINKER ($80), DEBUG I ($80), QED ($150), ZEDIT ($50), TOP I ($80), BASIC I ($50) and BASIC II ($100)

$815 Value **NOW $250**

CONFIGURATOR II

Includes: TPM-II ($125), Sample PIOS (BIOS) SOURCE ($FREE), MACRO II ($100), MACRO III ($150), LINKER ($80), DEBUG I ($80), DEBUG II ($100), QSAL ($200), QED ($150), ZTEL ($80), TOP II ($100), BUSINESS BASIC ($200) and MODEM SOURCE ($40) and DISASSEMBLER ($80)

$1485 Value **NOW $400**

MODEL I PROGRAMMER

This package is only for the TRS-80 Model I. Note: These are the ONLY CDL programs available for the Model I. It includes: TPM I ($80), BUSINESS BASIC ($200), MACRO I ($80), DEBUG I ($80), ZDDT ($40), ZTEL ($80), TOP I ($80) and MODEM ($40)

$680 Value **NOW $175**

MODEL II PROGRAMMER

This package is only for the TRS-80 Model II. It includes: TPM-II ($125), BUSINESS BASIC ($200), MACRO II ($100), MACRO III ($150), LINKER ($80), DEBUG I ($80), DEBUG II ($100), QED ($150), ZTEL ($80), TOP II ($100), ZDDT ($40), ZAPPLE SOURCE ($80), MODEM ($40), MODEM SOURCE ($40) and DISASSEMBLER ($80)

$1445 Value **NOW $375**

BASIC I - $50, a 12K+ basic interpreter with 7 digit precision.

BASIC II - $100, A 12 digit precision version of Basic I.

BUSINESS BASIC - $200, A full disk extended basic with random or sequential disk file handling and 12 digit precision (even for TRIG functions). Also includes PRIVACY command to protect source code, fixed and variable record lengths, simultaneous access to multiple disk files, global editing, and more!

ACCOUNTING PACKAGE - $300, Written in Business Basic. Includes General Ledger, Accounts Receivable/Payable, and Payroll. Set up for Hazeltine 1500 terminal. Minor modifications needed for other terminals. Provided in unprotected source form.

MACRO I - $80, A Z80/8080 assembler which uses CDL/TDL mnemonics. Handles MACROs and generates relocateable code. Includes 14 conditionals, 16 listing controls, 54 pseudo-ops, 11 arithmetic/logical ops, local and global symbols, linkable module generation, and more!

MACRO II - $100, An improved version of Macro I with expanded linking capabilities and more listing options. Also internal code has been greatly improved for faster more reliable operation.

MACRO III - $150, An enchanced version of Macro II. Internal buffers have been increased to achieve a significant improvement in speed of assembly. Additional features include line numbers, cross reference, compressed PRN files, form feeds, page parity, additional pseudo-ops, internal setting of time and date, and expanded assembly-time data entry.

DEVELOPER I

Includes: MACRO I ($80), DEBUG I ($80), ZEDIT ($50), TOP I ($80), BASIC I ($50) and BASIC II ($100)

$440 Value **NOW $150**

DEVELOPER II

Includes: MACRO II ($100), MACRO III ($150), LINKER ($80), DEBUG I ($80), DEBUG II ($100), BUSINESS BASIC ($200), QED ($150), TOP II ($100), ZDDT ($40), ZAPPLE SOURCE ($80), MODEM SOURCE ($40), ZTEL ($80), and DISASSEMBLER ($80).

$1280 Value **NOW $350**

DEVELOPER III

Includes: QSAL ($200), QED ($150), BUSINESS BASIC ($200), ZTEL ($80) and TOP II ($100)

$730 Value **NOW $300**

COMBO

Includes: DEVELOPER II ($1280), ACCOUNTING PACKAGE ($300), QSAL ($200) and 6502X ($150)

$1930 Value **NOW $500**

LINKER - $80. A linking loader for handling the linkable modules created by the above assemblers.

DEBUG I - $80. A tool for debugging Z80 or 8080 code. Disassembles to CDL/TDL mnemonics compatible with above assemblers. Traces code even through ROM. Commands include Calculate, Display, Examine, Fill, Goto, List, Mode, Open File, Put, Set Wait, Trace, and Search.

DEBUG II - $100. A superset of Debug I. Adds Instruction Interpreter, Radix change, Set Trap/Conditional display, Trace options, and Zap FCB.

6502X - $150. A 6502 cross assembler. Runs on the Z80 but assembles 6502 instructions into 6502 object code! Similar features as our Macro assemblers.

QSAL - $200. A SUPER FAST Z80 assembler. Up to 10 times faster than conventional assemblers. Directly generates code into memory in one pass but also to offset for execution in its own memory space. Pascal like structures: repeat...until, if...then...else, while...do, begin...end, case...of. Multiple statements per line, special register handling expressions, long symbol names, auto and modular assembly, and more! This one uses ZILOG Mnemonics.

QED - $150. A screen editor which is both FAST and easy to learn. Commands include block delete, copy, and move to a named file or within text, repeat previous command, change, locate, find at start of line, and numerous cursor and window movement functions. Works with any CRT having clear screen, addressable cursor, clear to end of line, clear to end of screen, and 80X24.

DISK FORMATS

When ordering software specify which disk format you would like.

CODE	DESCRIPTION
8SD	8" IBM 3740 Single Density (128 bytes/26 sectors/77 tracks)
8DD	8" Double Density (256 bytes/26 sectors/77 tracks)
8XD	8" CDL Extended Density (1024 bytes/8 sectcr/77 traceks 616K)
5SD	5.25" Single Density (TRS80 Model I, Versafloppy I, Tarbell I)
5EP	5.25" Epson Double Density
5PC	5.25" IBM PC Double Density
5XE	5.25" Xerox 820 Single Density
5OS	5.25" Osborne Single Density
5ZA	5.25" Z80 Apple (Softcard compatible)

TPM INFO When ordering TPM I or II, in addition to Disk Format, please specify one of the following codes:

	CODE	DESCRIPTION
TPM I:	NSSD/H	North Star Single Density for Horizon I/O
	NSSD/Z	North Star Single Density for Zapple I/O
	NSDD/H	North Star Double Density for Horizon I/O
	NSDD/Z	North Star Double Density for Zapple I/O
	TRS80-I	TRS-80 Model I (4200H Offset)
	TRS80II	TRS-80 Model II
	VI8	Versafloppy I 8"
	VI5	Versafloppy I 5.25"
TPM-II:	VII8	Versafloppy II 8" (XD)
	VII5	Versafloppy II 5.25"
	TRS80II	TRS-80 Model II (XD)

ZTEL - $80. An extensive text editing language and editor modelled after DEC's TECO.

ZEDIT - $50. A mini text editor. Character/line oriented. Works well with hardcopy terminals and is easy to use. Includes macro command capability.

TOP I - $80. A Text Output Processor for formatting manuals, documents, etc. Interprets commands which are entered into the text by an editor. Commands include justify, page number, heading, subheading, centering, and more.

TOP II - $100. A superset of TOP I. Adds: embedded control characters in the file, page at a time printing, selected portion printing, include/merge files, form feed/CRLF option for paging, instant start up, and final page ejection.

ZDDT - $40. This is the disk version of our famous Zapple monitor. It will also load hex and relocatable files.

ZAPPLE SOURCE - $80. This is the source to the SMB ROM version of our famous Zapple monitor. It can be used to create your own custom version or as an example of the features of our assemblers. Must be assembled using one of our assemblers.

MODEM - A communication program for file transfer between systems or using a system as a terminal. Based on the user group version but modified to work with our SMB board or TRS-80 Models I or II. You must specify which version you want.

MODEM SOURCE - $40. For making your own custom version. Requires one of our Macro Assemblers.

DISASSEMBLER - $80. Does bulk disassembly of object files creating source files which can be assembled by one of our assemblers.

HARDWARE

S-100 — SMB II Bare Board $50. "System Monitor Board" for S-100 systems. 2 serial ports, 2 parallel ports, cassette interface, 4K memory (ROM, 2708 EPROM, 2114 RAM), and power on jump. When used with Zapple ROM below, it makes putting a S-100 system together a snap.
Zapple ROM $35. Properly initializes SMB I/II hardware, provides a powerful debug monitor.

IBM PC — Big Blue Z80 board $595. Add Z80 capability to your IBM Personal Computer. Runs CP/M programs but does not require CP/M or TPM. Complete with Z80 CPU, 64K add on memory, serial port, parallel port, time and date clock with battery backup, hard disk interface, and software to attach to PC DOS and transfer programs. Mfr'd by QCS.
50% Discount on all CDL software ordered at the same time as a Big Blue (and for the Big Blue).

APPLE II — Chairman Z80 $345. Add Z80 capability to your Apple II/II Plus computer. Runs CP/M programs with our more powerful TPM. Includes 64K memory add on (unlike the competition this is also useable by the 6502/DOS as well as the Z80), TPM, QSAL assembler, QED Screen Editor, and Business Basic. Mfr'd by AMT Research.
Apple Special $175. Buy the Apple Z80 Developer at the same time as the "Chairman" and pay only $175 instead of $325.

APPLE Z80 DEVELOPER

Includes: 6502X ($150), MACRO II ($100), MACRO III ($150), QSAL ($200), QED ($150), LINKER ($80), DEBUG I ($80), DEBUG II ($100), ZDDT ($40) and BUSINESS BASIC ($200)

VALUE: $1250 **NOW $325**

$175 when purchased with AMT "Chairman" Board

ORDERING INFORMATION:

VISA/MasterCard/C.O.D.
Call or Write With Ordering Information....

OEMS:

Many CDL products are available for licensing to OEM's. Write to Carl Galletti with your requirements.

Dealer Inquiries Invited.

For Phone Orders ONLY Call Toll Free...

1-(800) 458-3491 (Except Pa.)

Ask For Extension #15

For information and Tech Queries call
(609) 599-2146

Prices and Specifications subject to change without notice.
TPM, Z80, CP/M, TRS80 are trademarks of CDL, Zilog, DRI and Tandy respectively.

Computer Design Labs
342 Columbus Avenue/Trenton, NJ 08629

Circle no. 12 on reader service card.

Virtual Personal Computer

Listing Two

(Listing continued, text begins on page 32)

```
#include < stdio.h>
#include < sgtty.h>
#include < signal.h>

/*
*
*       convert 7 bit even parity format (c7e)
*       to 8 bit mux format
*
*/

int ttyflg = 0;

struct sgttyb savtty;
struct sgttyb newtty;

/*
*
*       even parity table
*
*/

char parity[] = {
        0x00,0x80,0x80,0x00,0x80,0x00,0x00,0x80,
        0x80,0x00,0x00,0x80,0x00,0x80,0x80,0x00,
        0x80,0x00,0x00,0x80,0x00,0x80,0x80,0x00,
        0x00,0x80,0x80,0x00,0x80,0x00,0x00,0x80,
        0x80,0x00,0x00,0x80,0x00,0x80,0x80,0x00,
        0x00,0x80,0x80,0x00,0x80,0x00,0x00,0x80,
        0x00,0x80,0x80,0x00,0x80,0x00,0x00,0x80,
        0x80,0x00,0x00,0x80,0x00,0x80,0x80,0x00,
        0x80,0x00,0x00,0x80,0x00,0x80,0x80,0x00,
        0x00,0x80,0x80,0x00,0x80,0x00,0x00,0x80,
        0x00,0x80,0x80,0x00,0x80,0x00,0x00,0x80,
        0x80,0x00,0x00,0x80,0x00,0x80,0x80,0x00,
        0x00,0x80,0x80,0x00,0x80,0x00,0x00,0x80,
        0x80,0x00,0x00,0x80,0x00,0x80,0x80,0x00,
        0x80,0x00,0x00,0x80,0x00,0x80,0x80,0x00,
        0x00,0x80,0x80,0x00,0x80,0x00,0x00,0x80
};


/*
*
*       8 bit characters are read
*       the lower 7 bits contain data and the upper
*       bit has even parity set (hopefully)
*
*       parity is tested and if an error occurs
*       the character is dropped
*
*/

main(argc,argv,envp)
int argc;
char *argv[];
char *envp[];
```

```
{
        int c;
        int data;
        int resexit();

        init();
        signal(SIGINT,&resexit);
        signal(SIGHUP,&resexit);
        signal(SIGPIPE,&resexit);
        while((c=getchar()) != EOF){
                data = c & 0x7f;
                if(parity[data] == ((char)(c&0x80))){
                        put_mux(data);
                }
                else{
                        fprintf(stderr,"%s: parity error %o\n",argv[0],c);
                }
        }
        restore();
}

/*
*
*       init the tty if needed
*
*/
```

(Continued on next page)

Virtual Personal Computer

Listing Two

(Listing continued, text begins on page 32)

```
init()
{
        if(isatty(0) == 1){
            ttyflg = 1;
            gtty(0,&savtty);
            gtty(0,&newtty);
            /*      echo implicitly off   */
            newtty.sg_flags |= RAW;
            newtty.sg_flags &= ~ECHO;
            stty(0,&newtty);
            setbuf(stdin,NULL);
            setbuf(stdout,NULL);
        }
}

/*
*
*       send to mux
*
*       convert 7 bit bytes
*       to 8 bit format
*
*       7 bit format
*
*       3 bit base + 8    XXX01000
*       6.45 bit offset + 0xXXXXXX
*                         ----------
*       8 bit result      XXXXXXXX
*
*       the 3 bit base is encoded
*       as numbers in the range
*       32 to 39
*
*       the extra offset of 8 is not
*       encoded but is assumed as
*       shown above
*
*       the 6.45 bit offset is encoded as
*       numbers in the range 40 to 127
*
*       the 3 bit base is only changed
*       when necessary
*
*       if consecutive 8 bit numbers
*       fall in the same range then the
*       3 bit base is "shared" and
*       each 8 bit byte will only require
*       1 byte containing a 6.45 bit offset
*
*       the codes 0 to 31 are never enountered
*       so that control codes can be used
*       for flow control and other purposes
*
*       the codes 128 to 255 are not used
*       the high order bit is available for
```

(Continued on page 54)

SUBVERSIVE SOFTWARE

So cheap and useful it's...dangerous!

A radical idea in software development.
Useful Pascal programs, with:

- Complete annotated *source listings;*
- Disk(s) with *source code and compiled code;*
- User manual;
- Complete programmer documentation describing data structures and algorithms; and giving suggestions for modification.

Modify them, include them in your own systems, or simply use them.

A growing library of software tools you'll find hard to resist.

SUBVERSIVE SOFTWARE

TPL

The Text Processing Language. A text-file runoff program consisting of a set of text-processing primitive commands from which more complex commands (macros) can be built (as in Logo). Features include:

- Complete customization of text processing through macro definition and expansion, looping structures, and conditional statements;
- Adapts to any printer;
- Pagination;
- Text justification and centering;
- Indexing and tables of contents;
- Superscripts and subscripts;
- Bolding and underlining;
- Multiple headers and footers;
- End notes and footnotes;
- Widow and orphan suppression;
- Floating tables and 'keeps.'

$50

SUBVERSIVE SOFTWARE

DBX

Blocked Keyed Data Access Module. Maintains disk files of keyed data. Can be used for bibliographies, glossaries, multi-key data base construction, and many other applications.

- Variable-length keys;
- Variable-length data;
- Sequential access and rapid keyed access;
- Single disk access per operation (store, find, delete) in most cases;
- Multiple files;
- Dynamic memory allocation for RAM-resident index and current "page" of entries;
- Includes demonstration program and testbed program.

$50

SUBVERSIVE SOFTWARE

PDMS

The Pascal Data Management System. A user-oriented data management system in which numeric and alphanumeric data are stored in tables with named columns and numbered rows. Currently being used for dozens of different kinds of business and scientific applications, from inventory management to laboratory data analysis. Includes over 20 Pascal programs; more than 10,000 lines of code. Main features include:

- Maximum of 32,767 rows per file;
- Maximum of 400 characters per row, and 40 columns per table;
- Full-screen editing of rows and columns, with scrolling, windowing, global search/replace, and other editing features;
- Sorting, copying, merging, and reducing routines;
- Mailing label program;
- Reporting program generates reports with control breaks, totals and subtotals, and selects rows by field value; many other reporting features;
- Cross-tabulation, correlations, and multiple regression;
- Video-display-handling module;
- Disk-file-handling module.

Many other features. UCSD formats only.

$250

SUBVERSIVE SOFTWARE

ZED

Full-screen text editor; designed to be used either with TPL or by itself.

- Full cursor control;
- Insert mode with word wrap;
- 'Paint' mode;
- Single-keystroke or dual-keystroke commands;
- Command synonyms;
- Global search and replace;
- Block move, block copy, and block delete.

$50

BVERSIVE FTWARE

SCINTILLA

A log logit curve fitting program for radio-immunologic data; must be used with PDMS (described above).

- Multiple protocol files;
- Quality control files;
- Four-parameter non-linear curve fit.

UCSD formats only. **$250**

SUBVERSIVE SOFTWARE

CHROME

Chromatography data analysis program:

- Graphic display of analog data;
- Panning and zooming;
- Automatic peak-finding and baseline calculation;
- Full interactive peak editing;
- Computation of peak areas;
- Strip charts on C. Itoh and EPSON printers.

$100

SUBVERSIVE SOFTWARE

PLANE

Planimetry program:

- Bit-pad entry of cross sections;
- Real-time turtlegraphics display;
- Calculation of areas;
- Saves calculations to text file.

$100

VERSIVE TWARE

MINT

A terminal emulation program for communication between computers of any size.

- User-configurable uploading and downloading of files;
- X-ON/X-OFF and EOB/ACK protocols;
- Interrupt-driven serial input (for Prometheus Versacard in Apple II);
- Printer-logging.

$50

For more information, call 919-942-1411. To order, use form below or call our toll-free number: 1-800-XPASCAL

Check appropriate boxes:

FORMAT

□ 8″ UCSD SSSD
□ 5¼″ Apple Pascal
□ 5¼″ UCSD IBM PC 320k
□ 8″ CP/M SSSD
□ 5¼″ IBM MS-DOS
□ 5¼″ CP/M Osborne

PRODUCT	PRICE
□ DBX	$ 50
□ PDMS	$250
□ TPL	$ 50
□ ZED	$ 50
□ MINT	$ 50
□ SCINTILLA	$250
□ CHROME	$100
□ PLANE	$100

Name ______________________________

Address ______________________________

□ MasterCard □ VISA □ Check □ C.O.D.

(Please include card # and expiration date)

SUBVERSIVE SOFTWARE A division of Pascal & Associates,
135 East Rosemary St., Chapel Hill, NC 27514

Apple and Apple Pascal are trademarks of the APPLE Computer Corp. IBM and IBM PC are trademarks of International Business Machines. UCSD Pascal is a trademark of the Regents of the University of California. Osborne is a trademark of Osborne Computer. EPSON is a trademark of EPSON America, Inc. C. Itoh is a trademark of C. Itoh Electronics.

Circle no. 54 on reader service card.

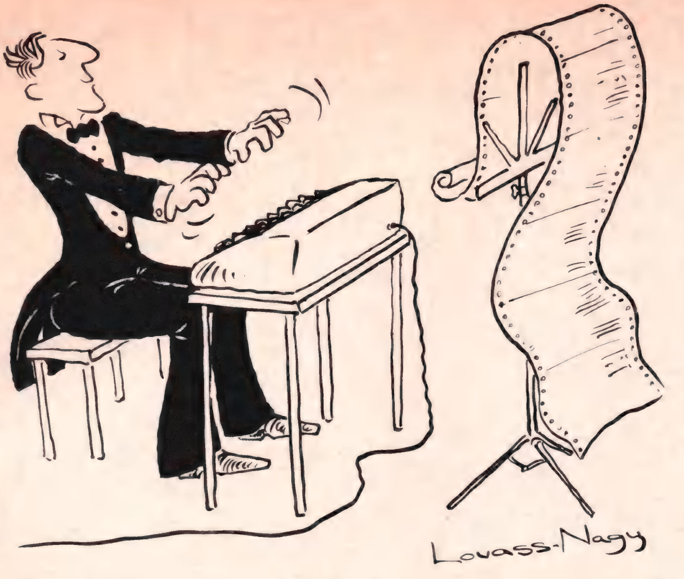
Virtual Personal Computer
Listing Two

(Listing continued, text begins on page 32)

```
*       a parity bit if needed
*
*/

put_mux(c)
int c;
{
        static int lbase = 0;
        char byte;

        byte = c & 0x7f;
        if(byte >= 32){
                if(byte <  40){
                        lbase = ((byte-32) << 5) + 8;
                }
                else{
                        putchar((byte-40)+lbase);
                }
        }
}

/*
*
*       restore and exit
*
*/

resexit()
{
        restore();
        exit();
}

/*
*
*       restore the tty
*
*/

restore()
{
        if(ttyflg == 1){
                stty(0,&savtty);
        }
}
```

End Listing Two

(Listing Three begins on page 56)



•NEW PRODUCTS•

Before Johann Sebastian Bach developed a new method of tuning, you had to change instruments practically every time you wanted to change keys. Very difficult.

Before Avocet introduced its family of cross-assemblers, developing micro-processor software was much the same. You needed a separate development system for practically every type of processor. Very difficult and very expensive.

But with Avocet's cross-assemblers, a single computer can develop software for virtually any microprocessor! Does that put us in a league with Bach? You decide.

The Well-Tempered Cross-Assembler

Development Tools That Work

Avocet cross-assemblers are fast, reliable and user-proven in over 3 years of actual use. Ask NASA, IBM, XEROX or the hundreds of other organizations that use them. Every time you see a new microprocessor-based product, there's a good chance it was developed with Avocet cross-assemblers.

Avocet cross-assemblers are easy to use. They run on any computer with CP/M* and process assembly language for the most popular microprocessor families.

5¼" disk formats available at no extra cost include Osborne, Xerox, H-P, IBM PC, Kaypro, North Star, Zenith, Televideo, Otrona, DEC.

Turn Your Computer Into A Complete Development System

Of course, there's more. Avocet has the tools you need from start to finish to enter, assemble and test your software and finally cast it in EPROM:

Text Editor VEDIT -- full-screen text editor by CompuView. Makes source code entry a snap. Full-screen text editing, plus TECO-like macro facility for repetitive tasks. Pre-configured for over 40 terminals and personal computers as well as in user-configurable form.

CP/M-80 version	$150
CP/M-86 or MDOS version	$195

(when ordered with any Avocet product)

EPROM Programmer -- Model 7128 EPROM Programmer by GTek programs most EPROMS without the need for personality modules. Self-contained power supply ... accepts ASCII commands and data from any computer through RS 232 serial interface. Cross-assembler hex object files can be down-loaded directly. Commands include verify and read, as well as partial programming.

PROM types supported: 2508, 2758, 2516, 2716, 2532, 2732, 2732A, 27C32, MCM8766, 2564, 2764, 27C64, 27128, 8748, 8741, 8749, 8742, 8751, 8755, plus Seeq and Xicor EEPROMS.

Avocet Cross-assembler	Target Microprocessor	CP/M-80 Version	•CP/M-86 IBM PC, MSDOS** Versions•
•XASMZ80	Z-80		
•XASM85	8085		
XASM05	6805	$200.00 each	$250.00 each
XASM09	6809		
XASM18	1802		
XASM48	8048/8041		
XASM51	8051		
XASM65	6502		
XASM68	6800/01		
XASMZ8	Z8		
XASMF8	F8/3870		$300.00 each
XASM400	COP400		
XASM75	NEC 7500	$500.00	
Coming soon: XASM68K...68000			

(Upgrade kits will be available for new PROM types as they are introduced.)

Programmer $389
Options include:
• Software Driver Package --
• enhanced features, no installation
• required.
• CP/M-80 Version .... $ 75
• IBM PC Version .... $ 95
RS 232 Cable .... $ 30
8748 family socket adaptor ... $ 98
8751 family socket adaptor ... $174
• 8755 family socket adaptor ... $135

• **G7228 Programmer by GTek** -- baud
• to 2400 ... superfast, adaptive program-
• ming algorithms ... programs 2764 in one
• minute.

• Programmer $499

• Ask us about Gang and PAL programmers.

• **HEXTRAN Universal HEX File Con-**
• **verter** -- Converts to and from Intel,
• Motorola, MOS Technology, Mostek,
• RCA, Fairchild, Tektronix, Texas
• Instruments and Binary formats.

• Converter, each version $250

Call Us

If you're thinking about development systems, call us for some straight talk. If we don't have what you need, we'll help you find out who does. If you like, we'll even talk about Bach.

CALL TOLL FREE 1-800-448-8500
(In the U.S. except Alaska and Hawaii)

VISA and Mastercard accepted. All popular disc formats now available -- please specify. Prices do not include shipping and handling -- call for exact quotes. OEM INQUIRIES INVITED.

*Trademark of Digital Research **Trademark of Microsoft

AVOCET SYSTEMS INC.™
DEPT.
804 SOUTH STATE STREET
DOVER, DELAWARE 19901
302-734-0151 TELEX 467210

Circle no. 4 on reader service card.

Virtual Personal Computer (Listing continued, text begins on page 32)

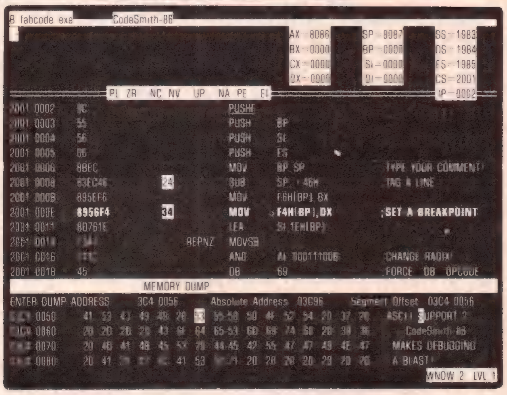
Listing Three

```
#include <stdio.h>

/*
*
*       multiplexor to piu conversion
*       level 0
*
*       an 8 bit stream is read from the standard input
*       and a 16 bit stream is written on the standard output
*
*       the 8 bit stream usually comes from the c78 converter or
*       other link level processors
*
*       the 16 bit stream that is written on the standard output
*       is created by writing 2 bytes
*       the low byte is written first followed by the high byte
*
*       the 16 bit channel is usually sent to a PIU or VDS module
*
*/

unsigned base = 0;

/*
*
*       mux input
*
*       break down as follows:
*
*               0XXXXXXX - add base to XXXXXXX yielding 16 bit data
*               10XXXXXX - set middle 6 bits of base
*               110XXXXX - set upper 5 bits of base
*               111XXXXX - 5 bit remote control channel value
*
*       the 5 bit remote control channel is dumped because this
*       is a level 0 VPC utility
*
*/

main()
{
        int c;
        int data;

        setbuf(stdin,NULL);
        setbuf(stdout,NULL);
        while((c=getchar()) != EOF){
                if(c <  128){
                        data = base + (c&0x7f);
                        put_piu(data);
                }
                else{
                        if(c <  192){
                                base &= 0xf81f;
                                base |= (c&0x3f) << 5;
                       }
                       else{
                               if(c <  224){
```

```
                                        base &= 0x07ff;
                                        base |= (c&0x1f) << 11;
                                }
                                else{
                                        /*      eat the control channel */
                                }
                        }
                }
        }
}

/*
*
*       output to piu
*
*/

put_piu(data)
int data;
{
        putchar(data & 0xff);
        putchar((data>>8)&0xff);
}
```

End Listing Three

(Listing Four begins on page 58)



```
B fabcode exe          CodeSmith-86
                                              AX=8086   SP=8087   SS=1983
                                              BX=0000   BP=0000   DS=1984
                                              CX=0000   SI=0000   ES=1985
                                              DX=0000   DI=0000   CS=2001
              PL ZR  NC NV  UP  NA PE  EI                         IP=0002
2001 0002  9C                       PUSHF
2001 0003  55                       PUSH   BP
2001 0004  56                       PUSH   SI
2001 0005  06                       PUSH   ES
2001 0006  8BEC                     MOV    BP SP        (TYPE YOUR COMMENT)
2001 0008  83EC46      24           SUB    SP 46H       TAG A LINE
2001 000B  895EF6                   MOV    F6H[BP] BX
2001 000E  8956F4      34           MOV    F4H[BP],DX   ;SET A BREAKPOINT
2001 0011  8D761E                   LEA    SI 1EH[BP]
2001 0014  F3A4            REPNZ    MOVSB
2001 0016  [illegible]              AND    AL 00011100B  CHANGE RADIX
2001 0018  45                       DB     69           FORCE  DB  OPCODE
                       MEMORY DUMP
ENTER DUMP ADDRESS   3C4 0056    Absolute Address  03C96    Segment Offset  03C4 0056
[illegible] 0050  41 53 43 49 49 20 53 55 50 50 4F 52 54 20 32 20   ASCII SUPPORT 2
[illegible] 0060  20 20 20 20 43 6F 64 65 53 6D 69 74 68 20 38 36     CodeSmith-86
[illegible] 0070  20 4D 41 4B 45 53 20 44 45 42 55 47 47 49 4E 47    MAKES DEBUGGING
[illegible] 0080  20 41 [illegible] 41 53 [illegible] 20 20 20 20 20 20 20   A BLAST!
                                                              WNDW 2  LVL 1
```

Virtual Personal Computer

(Listing continued, text begins on page 32)

Listing Four

```
#include < stdio.h>

/*
*
*       piu to multiplexor converter
*       level 0
*
*       this program reads the standard input and converts a 16 bit
*       stream to multiplexor format
*       the standard input contains 16 bit values encoded as 2
*       bytes, low byte followed by high byte
*
*       the standard output is used and contains data in 8 bit
*       multiplexor format
*       the output is usually directed to a link level process
*
*/

unsigned base = 0;

main()
{
        int low;
        int high;

        setbuf(stdin,NULL);
        setbuf(stdout,NULL);
        while((low=getchar()) != EOF){
                high = getchar();
                put_mux(((high & 0xff) << 8) + (low & 0xff));
        }
}

/*
*
*       send 16 bit data value through mux
*
*       there are other clever ways of setting the base values
*       to yield better link performance
*       this method is easy to program and describe
*
*       if the value to be sent can be reached using the current
*       base then only a 7 bit offset is sent
*       if the value can not be reached, a new base is
*       determined by the upper 11 bits of the current value
*       the 5 and 6 bit sub-base registers are compared with the
*       current base register and one or both are changed as needed
*       the value is then sent using this new base
*
*/

put_mux(value)
unsigned value;
{
        unsigned diff;
        unsigned new_bas;

        if((diff = (value - base)) < 128){
                putchar(diff);
        }
        else{
                new_bas = value & 0xffe0;
                if((new_bas & 0xf800) != (base & 0xf800)){
                        putchar(((new_bas & 0xf800) >> 11) | 0xc0);
                }
                if((new_bas & 0x07e0) != (base & 0x07e0)){
                        putchar(((new_bas & 0x07e0) >> 5) | 0x80);
                }
                putchar(value & 0x1f);
                base = new_bas;
        }
}
```

End Listing Four

PABX and the Personal Computer

You are probably wondering why a journal like *Dr. Dobb's* is discussing phone systems. Well, the private area branch exchange (PABX) is now a prime candidate for use as a local area network (LAN) for computer connections. Let's take a look at what a PABX is and see if the PABX is really the LAN of the future or simply a way to get rid of some wire in a building.

What's a PABX?

The PABX is a local telephone system usually limited to a building, office, campus, or geographical area. Its purpose is to replace the "expensive" facilities of the telephone company with equipment owned or leased by the business itself to save money and improve communications. PABX systems are provided by common carriers such as the local Bell Telephone system or by independent suppliers such as Northern Telecom, Intecom, and others. Offices acquired PABXes because of a need to communicate within an organization without sending all calls through the local telephone system's central office switching facility.

State-of-the-art PABX systems have a number of distinguishing features that make them prime candidates for data movement media. One such feature is the digital nature of the switching facility. When you pick up the receiver on your digital telephone and "dial" a number, a connection is made at the PABX by routing the call through a digital switching circuit, usually via a time-division multiplexing matrix switch. When you talk, your voice is converted from an analog signal into a digital signal at a voice transmission rate of 32K or 64K bits per second. If the connection is within the building or switching area of the PABX, the signal remains digital. If the connection is to an outside facility, i.e., for a long distance call, the digital voice signal is again converted to an analog signal and placed on the common carrier trunk (the wires of the phone company).

Your digital phone set is an electronic marvel (see Figure 1, at right). It has a microprocessor built in to keep track of your speed-dialing numbers, the last number you dialed, and other features that are found on the friendly faceplate. It also controls the A/D conversion of your voice. Connecting your phone to the PABX are two pairs of wires: a power pair and a signal pair. The signal pair is used to transfer voice information and, in most cases, digital data traffic as well.

by Ted Rohling

Ted Rohling, 4041 Medical Drive, San Antonio, TX 78229.

The Computer Connection

Every communications network has a topology, that is, the way the wires are connected together. The PABX uses a star network, one where stations are connected in radials from a central point of control (see Figure 2 on page 61). The first telephone system was one of the first star-configured communications networks. Then the mainframe computer came along and adopted the same structure for its computer-to-terminal connections. In fact, one of the first uses of a PABX as a LAN was in a mainframe/terminal interconnect. Why is the PABX a better solution than those available in the past?

For one thing, utilizing the PABX as an interconnect device gets rid of all of the twisted-pair wire that is usually strung around the building for data traffic. Simply add an RS-232 or RS-449 connector to the back of the phone set, include more logic in the phone and the switch used to handle the data traffic, and connect your terminal. Now you can use your phone to dial a computer connection. The PABX is also connected to the computer terminal port so that a "conversation" can take place between the terminal and the computer (see Figure 3, page 62).

Using the PABX also reduces the number of terminal ports required on the mainframe or minicomputer. Computer users are like phone users: not all of them use the computer all day. In a large office a limited number of outside telephone lines are usually shared by all of the workers. Those people that need to have regular outside access usually have a dedicated phone line for themselves. This keeps down the cost of the telephone service. The same applies to computer connections. In most installations casual users come and go; their hard-wired connections are wasted for a given part of the day (see Figure 4 on page 62). By connecting a certain number of terminal ports for casual users through the PABX, you can reduce the port configuration (see Figure 5 on page 63). When the user dials the computer, the PABX looks for the next available port and makes the connection. When a terminal is disconnected, a port is freed for other users to access.

The Micro Connection

LANs typically move data from one point to another using some common communications capability. The network allows the data to be looked at as either complete files that are moved around from workstation to workstation for pro-

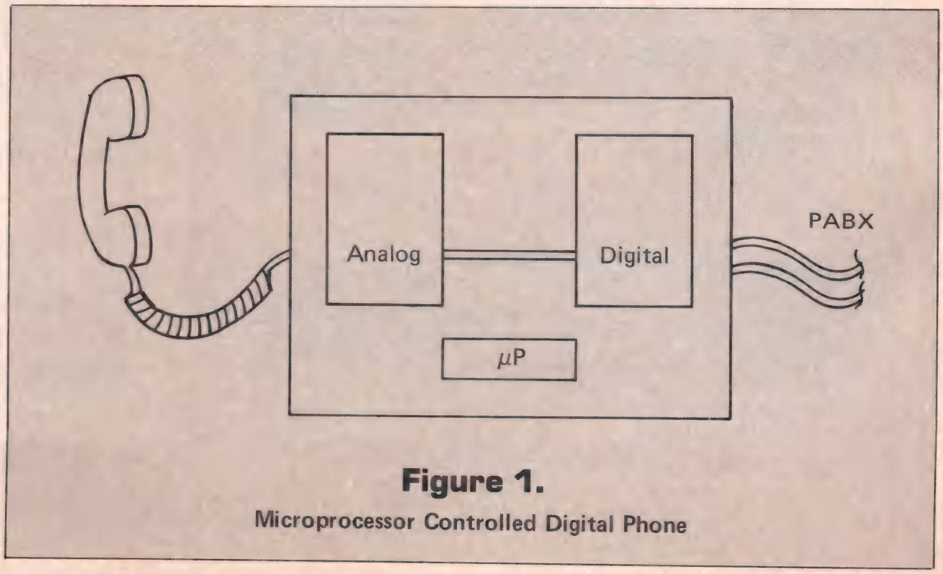


Figure 1.
Microprocessor Controlled Digital Phone

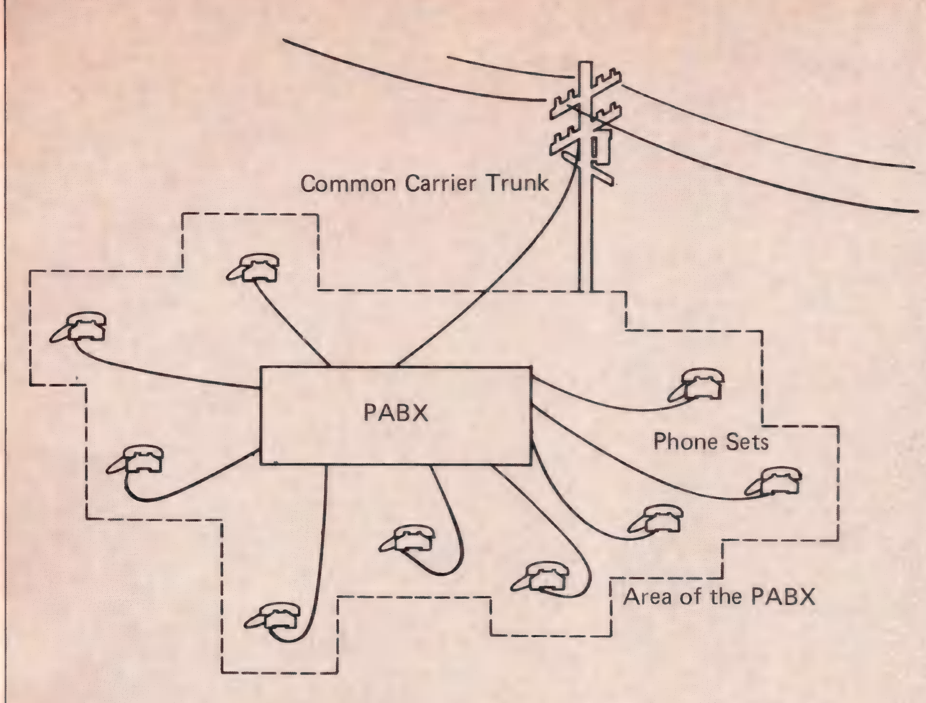


Figure 2.

PABX as a STAR Network

cessing or as "phantom disk drives." When using a LAN with phantom drives, you don't have to move the complete file to your workstation to process it. You can use the file where it is located, retrieving and updating records as if they were on your local disk devices. Certain networks utilizing coaxial cable and high-speed links can provide phantom drives.

Microcomputers in a PABX LAN installation are normally used in a file transfer or an attached terminal mode. The connections between micros within the scope of a PABX are simplified when a digital switch is used. The modems are replaced by the digital phones, and the phone connection is made as usual. The advantage is that a micro can now be moved from location to location without dragging the modem around with it.

Speed also is improved between 800% and 1600%, depending on the PABX utilized: data rates of 9.6 kilobaud or 19.2 kilobaud are common, which is a big improvement over the usual 300- or 12-baud modem used in dialup communications today. Moving a 20 kilobyte file at 300 baud under ideal circumstances takes over 11 minutes; at 9600 baud it takes about 21 seconds. These times obviously discard any disk I/O time or line error problems.

Is the PABX really a cost-savings approach to the local area networks? If a new installation is being developed, the answer is yes. Since today's average cost per telephone station for a digital PABX installation is $1000, the addition of the digital data capability costs about the same as a low-cost medium-speed modem. However, to completely retrofit an existing PABX with digital data capabilities simply for the LAN facility is not an economically feasible solution. Nor is the PABX a solution for small businesses, compared to the relatively low cost of "key systems."

Conclusion

Utilizing the PABX for a local area network in the microcomputer environment is not an optimum solution for most installations. Other network capabilities, including those now being defined by IEEE 802 committees or those on the market such as ARCNET or ETHERNET, are more applicable to the microcomputer world. However, for locations where digital PABX facilities are available, the users can now enjoy faster file transfers between their microcomputers and can connect their micro to the mainframe or minicomputer systems at much higher data rates. DDJ

Reader Ballot
Vote for your favorite feature/article.
Circle Reader Service **No. 196.**

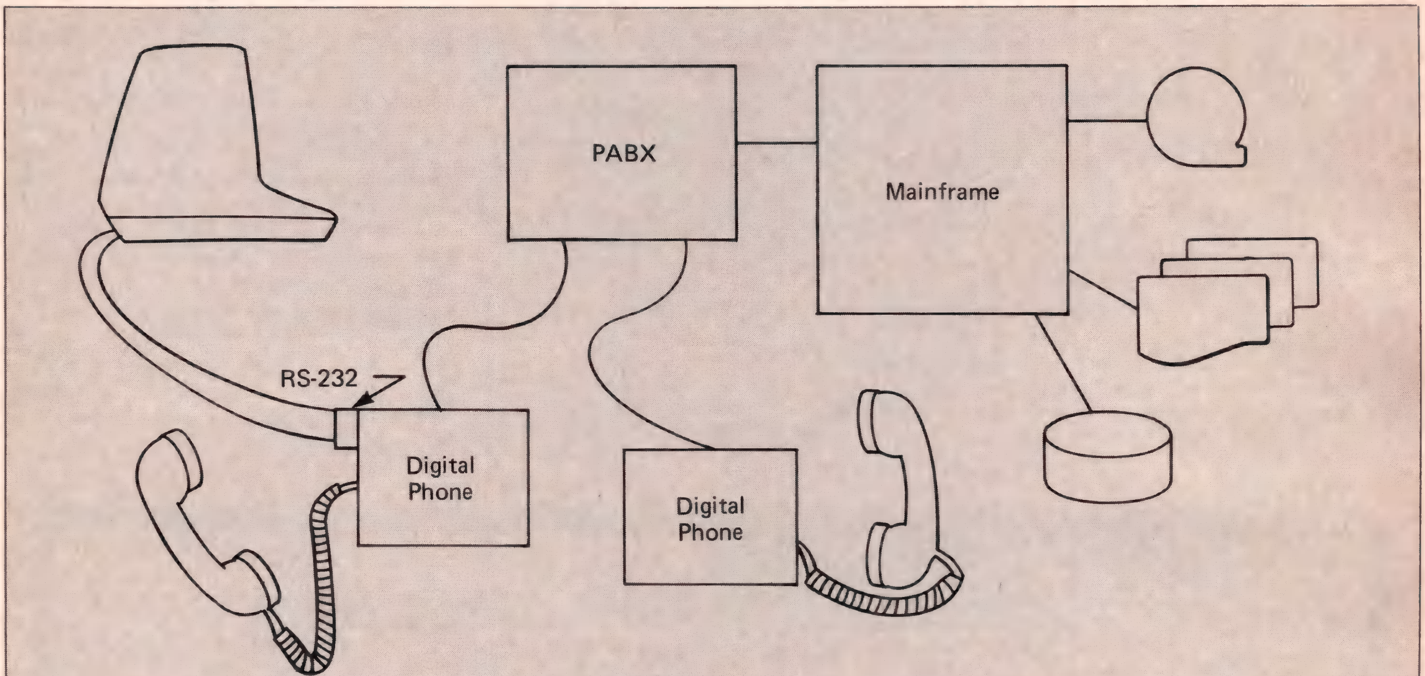


Figure 3.

PABX as a Data Interconnect Device as well as Voice Carrier

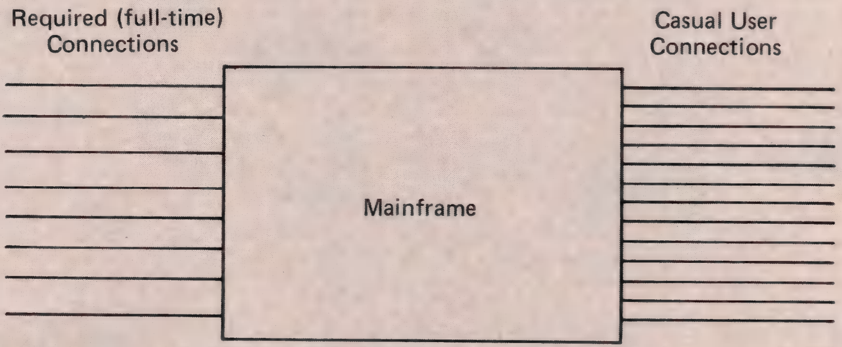


Figure 4.

Typical Configuration

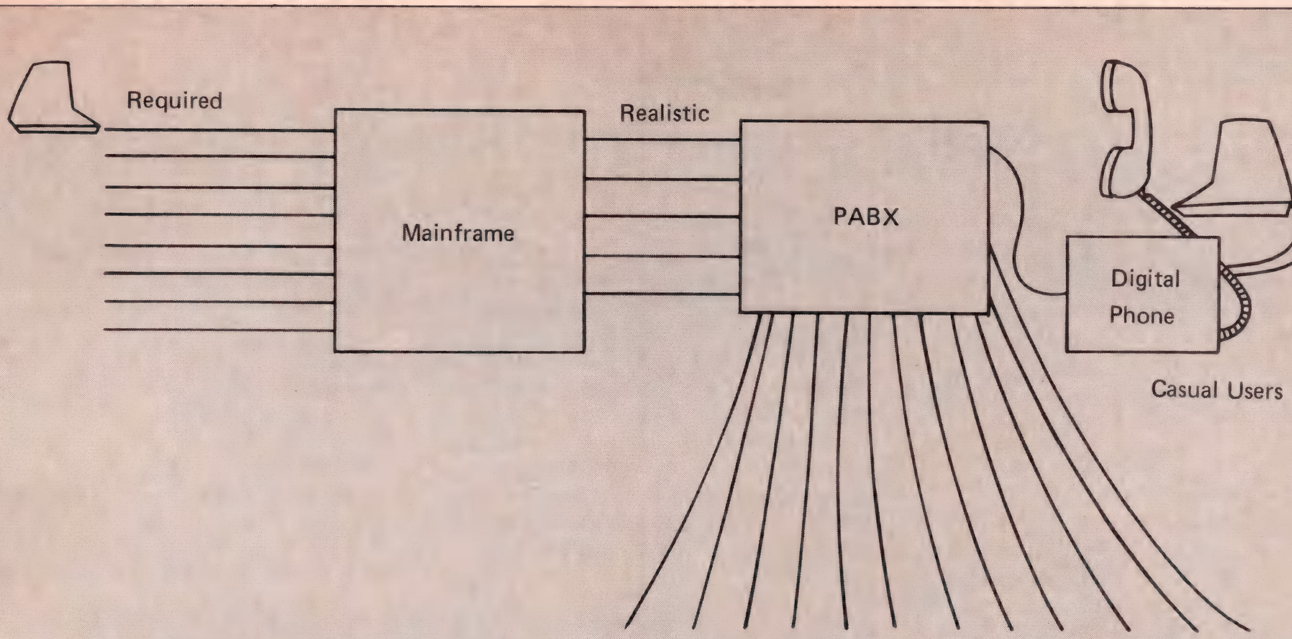


Figure 5.
Reduced Mainframe Ports Through PABX Connections

BASIC Language Telecommunications Programming

This article addresses some aspects of the following issues involving communications to a remote host (mainframe) computer:

- How to upload files to systems that will not let us install receive programs such as Modem7 protocol programs
- How to get BASIC to read the status of our microcomputer's hardware

The programs described in this article were developed to work with systems having these two limitations. First, we cannot install any programs on the host to help us communicate with it. The host is used, for example, as a mail drop or bulletin board system and only allows the computer to transfer ASCII text files; no binary format files may be transferred.

Second, we must not overload the host computer. Just about every mainframe computer that has keyboard or teletype (TTY) ports expects data to be transferred into that port at the typing speed of a human being. We, however, want to force-feed data into such a host's TTY port from our microcomputer; as a result, data will be lost when we overrun the ability of the host computer's operating system to absorb what we send. This sounds unbelievable, but it has occurred on every one of the dozen or so different mainframes we have tried this on. Some of these famous brands could not even accept data at a continuous rate of 300 baud (30 characters per second).

Echo Handshaking

The cure for all this is to sense the presence of an echo character coming back from the host and to use that to determine the timing of the whole system. This is called echo handshaking. This method is not limited by the baud rate of the modem being used: the transmission and echo times dominate the transmission speed.

The sequence goes something like this. Using some software means to directly connect our keyboard with the host through the printer/modem port, we prepare the host computer to accept data and to transfer it to a file in the host computer. We then start the echo handshaking communication program. The program sends the first character of the file up to the host and waits for the host to echo the character in normal full-duplex operation. When the program detects a returned character, it sends the next one, and so on until the entire file is transferred to the host. When the program is finished, we connect directly to the host again and terminate the transmission by whatever means the host wants.

We use the TELNET program, available on older BDS 'C' distribution diskettes, for the direct connection from keyboard to host through the CP/M LST: port on the microcomputer. This well-done piece of software allows the easy downloading of files from the host to the microcomputer.

The concepts in the BASIC programs shown in Listings One and Two (page 66 and 67, respectively) could be inserted into the TELNET program, but because we communicate with a wide variety of

by R. S. Broughton

Robert S. Broughton, P.O. Box 5191, Beaverton, OR 97006.

I/O READ PORTS:

	DATA	STATUS
Channel A, printer (LST:) port	80h	81h
Channel B, terminal (CON:) port	82h	83h

Figure 1.

I/O Register Addresses

Z80–SIO Receive Port Status Register:

D7	D6	D5	D4	D3	D2	D1	D0

Where the individual bits are defined as:

D7 – Break Abort
D6 – Transmit Underrun/EOM
D5 – Clear To Send
D4 – Sync/Hunt
D3 – Data Carrier Detect
D2 – Transmit Buffer Empty
D1 – Interrupt Pending
D0 – Receive Character Available

Figure 2.

Z80–SIO Status Register Description

host computers, we felt that a separate program for each protocol would suffice. We use these programs on a Cal-Tex BigBoard II microcomputer.

The program shown in Listing One will communicate with most Unix computer systems and bulletin board systems. For other computer hosts such as Hewlett-Packard (H-P) computers, however, a slightly different protocol is used. A program that uses this different protocol is shown in Listing Two.

The H-P computers allow data to be entered by using one of several editor programs (e.g., EDIT2 or EDITOR) after placing the program into insert mode. The protocol for these editors is slightly more complex than editors typical of the Unix systems. When in insert mode, the H-P editors send out a prompt for each line, which includes the current line number. This prompt ends with a control-Q (X-ON), which is the signal to begin entering the next line. After sending the carriage return for the line, the program in Listing Two discards characters received from the H-P host until the control-Q is received. The characters discarded are the prompt for the next line. The next line is then sent, character by character, each time waiting for the echo of the preceding character until the next character is sent, and so on.

Discussion

Since the programs are very similar, we will examine only the version for Unix computers in Listing One. The comments within the version for H-P computers in Listing Two should explain the protocol differences between the two. The statements are listed in Table I (below).

Statement 450 reads a byte from register 129 (decimal) or 81 (hex) by using the INP instruction. The AND 1 instruction masks or turns off all bits in this byte except for the low-order bit; on the BigBoard II microcomputer using the Zilog Z80-SIO serial I/O chip, this bit is the data-ready bit in the status register of the SIO. Statement 450 will loop on itself as long as this bit is a zero, which means no characters have been received by the serial I/O device. When a character has been received by the serial I/O unit, this status bit goes high, returning a one and allowing control to pass to the next statement.

100-130	Initialize the environment.
140-170	Enter dialog with user to gain the filename to transfer and open it as an input file.
200-360	The main body loop of the program.
370-400	Report statistics of the file transfer to the user and determine whether another file transfer is wanted.
450-470	A subroutine that checks the status register of the serial I/O device and reads a byte of data from the data register when a character has been received.

Table I
Statements

Statement 460 reads a byte from the data register of the Z80-SIO chip. The AND 127 statement masks or turns off the top bit of the byte just read, yielding a single 7-bit ASCII character into X$. This character will ultimately be displayed on the CRT of the local microcomputer, letting the user know what is being transmitted.

We don't have to do anything special to send data out the printer port to the modem; we just use the LPRINT statement that Microsoft BASIC provides. For reading the echo back from the printer port from the modem, a more complicated method is required. As you can imagine, very few systems would anticipate the desire to read data back from a printer.

No matter, most systems use a serial I/O device that has both a transmit and a receive section, and both sections are probably connected to the appropriate serial printer port. For our communication purposes, the serial printer port is connected to a modem and the modem to the telephone line. Discussion of the modem connection is beyond the scope of this article.

The BigBoard II microcomputer uses the Z80-SIO serial I/O chip for all serial I/O, both to/from the console terminal and to the printer. This discussion generally applies to other serial I/O chips or USARTs in use (i.e., the i8251, ns8250, etc.). In this particular configuration, I/O read address 80 (hex) is the I/O read address for channel A or the printer (LST:) port, with 82 (hex) being the I/O read address for channel B or the terminal (CON:) port. This is summarized in Figure 1 (page 64). The bits in the status register are described in Figure 2 (page 64).

What interests us here is bit D0, which is a one when a character has been received, indicating we are ready to read from the receive data register. So we loop on this bit until it turns from a zero into a one, at which time we read the character just received at the receive data register of the SIO.

As an example of how these programs might be changed to conform to other serial I/O devices, the i8251 has its Receive Character Available bit in D1, not D0. For this chip, you would AND the status byte read with two in order to set all bits read from the status register of the 8251 to zero (except bit D1). The IF statement here should test for <>2, not <>1.

There are several possibilities for enhancing the ideas expressed here. Error checking could be performed by comparing the return echo character with the original character. If they are different, there was an error in transmission or reception. If an error is detected, send a backspace and resend the original character. Of course, you won't know whether the error was in the link to the host or from the host to your microcomputer. To speed up transmission, try sending a space and then a backspace before each line so that transmission and echo can overlap. You'll also find that the techniques in this article will assist the direct transfer of data from one microcomputer to another, without a telephone modem hookup. Hopefully, these remarks and programs will be helpful to you as you telecommunicate with your microcomputer. DDJ

Reader Ballot
Vote for your favorite feature/article.
Circle Reader Service **No. 197.**

BASIC Telecommunications Programming (Text begins on page 64)

Listing One

```
100 REM "ToUnix" transfers data files to unix cpu running cat, from cp/m
110 REM                                 printed__________@________
120 WIDTH LPRINT 255             'no system carriage returns
130 TRUE=-1
----------------------------------------------------------

140 PRINT"ToUnix here"
150 PRINT"Enter file name to transfer ";
160 INPUT FL$
170 OPEN "I",1,FL$               'open the input file
----------------------------------------------------------

180 CHARS=0
190 LINES=0

----------------------------------------------------------

200 IF EOF(1) THEN 370           'check for end of input file
210 LINE INPUT #1,A$             'read next line from input file
220 IF LEN(A$)<1 THEN A$=" "     'if a null string send a space anyway
230 LINES=LINES+1                'count statistics
240 A=LEN(A$)
250 CHARS=CHARS+A                'count statistics
260 FOR I=1 TO A                 'for all the characters in the line
270   LPRINT MID$(A$,I,1);
280   GOSUB 450                  'wait for echo back from unix computer
290   PRINT CHR$(X);             'display it on the terminal
300 NEXT I
310 LPRINT CHR$(13);             'send the carriage return to end the line
320 GOSUB 450                    'wait for character received from modem
330 IF X<>10 THEN 320            'line feed received, the prompt for next line
340 LINES=LINES+1                'one more line sent
350 PRINT
360 GOTO 200                     'get next line from input file

----------------------------------------------------------

370 CLOSE
380 PRINT                        'print statistics of the transfer:
390 PRINT "File transferred"
400 PRINT CHARS;" Characters transferred"
410 PRINT LINES;" Lines transferred"
```

```
420 PRINT CHR$(7);              'beep to operator that transfer is completed
430 GOTO 150                    'get next file to transfer, if any
440 END
----------------------------------------------------------------------

450 IF((INP(129) AND 1) <>1) THEN 450 'wait for data to read back
460 X=INP(128) AND 127          'readback the echo from the host computer port
470 RETURN
```

End Listing One

Listing Two

```
100 REM "ToHP" Transfers text files to H-P computer running edit2,from cp/m
110 REM                                          PRINTED__________(d________
120 WIDTH LPRINT 255            'no system carriage returns
130 TRUE=-1
----------------------------------------------------------------------

140 PRINT"ToHP HERE"
150 PRINT"ENTER FILE NAME TO TRANSFER ";
160 INPUT FL$
170 OPEN "I",1,FL$
180 CHARS=0
```

(Continued on next page)

BASIC Telecommunications Programming

Listing Two (Listing continued, text begins on page 64)

```
190 IF EOF(1) THEN 350          'check for end of input file
200 LINE INPUT #1,A$            'read a line of data from input file
210 IF LEN(A$)<1 THEN A$=" "    'if a null string then send a space anyway
220 LINES=LINES+1
230 A=LEN(A$)                   'how many characters are in this line
240 CHARS=CHARS+A

-------------------------------------------------------------

245 LPRINT CHR$(17);            'send leading control Q...
250 FOR I=1 TO A                'for all the characters in the line
260   LPRINT MID$(A$,I,1);      'send the next character in the line
270   GOSUB 430                 'wait for echo back from the H-P computer
280   PRINT CHR$(X);            'display the echo on the local terminal
290 NEXT I
300 LPRINT CHR$(13);            'send the carriage return to end this line

-------------------------------------------------------------

310 GOSUB 430                   'wait for character to be returned from modem
320 IF X<>17 THEN 310           'discard characters until the control Q prompt
```

```
------------------------------------------------------------

330 PRINT                         'local carriage return
340 GOTO 190                      'get the next line, if any

------------------------------------------------------------

350 CLOSE                         'all done with this file,
360 PRINT                         'report status and statistics to user
370 PRINT"FILE TRANSFERRED"
380 PRINT CHARS;" CHARACTERS TRANSFERRED"
390 PRINT LINES;" LINES TRANSFERRED"
400 PRINT CHR$(7);
410 GOTO 150                      'try to get another file to transfer
420 END

------------------------------------------------------------

430 IF((INP(129) AND 1) <>1) THEN 430 'wait for data to echo back
440 X=INP(128) AND 127            'readback the echo from the H-P computer port
450 RETURN
```

End Listing

U.S. Robotics' S-100 Modem

We found this 1200-baud S-100 plug-in modem very interesting – it's the only one we know of. We reserved detailed examination of the product for an upcoming review, deciding instead to let someone intimately associated with the product give you a look at the process of its creation. – Ed.

About ten years ago I used my first modem. It was a large gray box in a drawer of a special desk holding an IBM Selectric-type terminal. This modem communicated with a large mainframe computer at the lightning speed of eleven characters per second. Since this replaced punched cards, the slow speed did not seem so bothersome. But one day a thirty-character-per-second modem was installed: this was high speed computing.

Ten years later, I am sitting at my own computer with a modem installed in the machine and I can communicate with other computers over the telephone network at over 120 characters per second. I never thought I would own a computer, let alone work for a company that manufactures modems.

The last ten years (for data communications, the last two or three) have seen tremendous advances in telecommunications software and hardware. Today, both the computer and the modem are priced within the reach of the average computer user.

Less than a year ago, U. S. Robotics decided to redesign the Autodial 212A unit which was the top of the line product produced by the company. The main goals of that effort were to produce a smaller, less expensive product and a simpler-to-use modem. This product would help to make 1200-baud intelligent modems more available to the average computer user.

Most of the auto dial units then on the market looked like many of the early home computers. They included many lights and switches. The average modem user, like today's average computer user, usually needs an on/off indicator and an on switch to turn the unit on and off. The lights are useful for real technicians, but most people could care less.

by Michael McKillip

Mike McKillip, U.S. Robotics, 477 E. Butterfield Rd., Lombard, IL 60148.

In addition to the simpler user interface, the design team developed the concept of a core modem. The core modem circuitry contained all the electrical modem elements for accepting TTL-level digital signals and producing analog signals necessary for telephone transmission. By adding a power supply, RS232 interface, and other circuitry, this product could be customized for other equipment manufacturers or made into specialized products.

Several of the members of the engineering team were long-time users of S-100 systems. After the production of the Password, the new redesigned modem, the attention of several people turned to other products to incorporate the core modem concept. Among the first was the S-100 modem. Although there were several S-100 modems available, there were no intelligent 1200-baud modems for S-100 computers.

Advantages and Disadvantages of the S-100 Modem

The S-100 card modem presents several advantages for the user. For people like me there is the question of space. A card modem is integral to the computer and does not take up any of the work area around the computer.

Cables, it seems to me, are the Gordian knot of computer owners. For most modems, at least three cables are needed: power, telephone line, and RS232 connector. The power for the modem comes from the S-100 bus, so there is no adaptor to plug into an already overcrowded power strip. Communication between the modem and the computer is over the S-100 bus which eliminates the RS232 cable. The S-100 modem requires only the telephone cable.

The elimination of the RS232 interface also removes the problem of cabling between the modem and computer. Removing the necessity for an external RS232 cable also eliminates the need to use one of the serial ports on the computer for communications.

Today many of the modems on the market are intelligent modems. These modems are much simpler to use than many of the earlier models. For the most part they are controlled by commands sent over the serial communications line. Commands may be incorporated to dial numbers, turn on and off speakers, send result codes, and numerous other functions. The commands provide a means for software control of all the modems' functions. In terms of control, the S-100 modem is the ultimate intelligent modem. There is no on/off switch; it is a fully integrated part of the computer.

Systems developers or integrators can develop programs which can establish a communications link without the user of the system ever knowing the telephone number dialed or log-on sequences required. This is possible through the use of the autodial function and direct input/output over the S-100 bus. Sophisticated users may be able to circumvent such techniques.

There are some disadvantages to the S-100 modem. Since it is not a separate unit, it cannot be easily moved from one machine to another like a stand-alone unit. It may be slightly more complicated to install the S-100 modem for the simpler types of communications. Most systems provide some driver routines for serial communications devices, but the S-100 may require user-written driver routines.

The decision to use an external modem or an integrated modem depends on the needs and preferences of the user more than on the units themselves.

The S-100 Modem Design Process

The design team started with the core modem, which they now refer to as "essence of modem," and began to design a card modem for S-100 systems. One of the first factors taken into consideration was the S-100 standards. The design team had had experience with S-100 cards in the past and knew the troubles with non-standard cards. The first requirement of the new design was that it be compatible with the new IEEE 696 standard for S-100 systems.

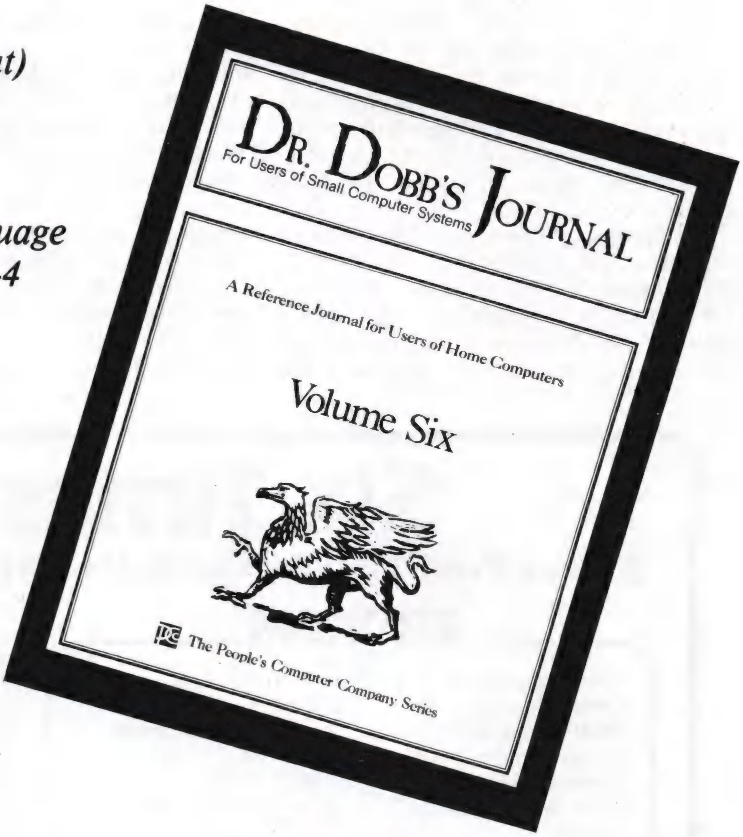
The bulk of the design work then centered on the serial interface. Several important criteria came into play when designing the interface. First was the availability of the parts to be used (were several sources available?); familiarity with serial interfaces (what experience did the designer have with the parts?); and simplicity of operation (would the user be able to operate the modem with a basic knowledge of S-100 systems?).

The 8251 USART (Universal Synchronous/Asynchronous Receiver/Transmitter) was chosen for the basic design because of availability of the part and the

Didn't get what you wanted for Christmas? Order Volume 6!

At last what you've been waiting for: ALL the issues from 1981 in one GIANT volume—over 550 pages long!

- *The first all-FORTH issue (now sold out)*
- *More Small-C development*
- *Continuing coverage of CP/M*
- *Santa Barbara Tiny BASIC for 6809*
- *Pidgin - A Systems Programming Language*
- *Write Your Own Compiler with META-4*
- *The first "Clinic" columns*
- *The Electric Phone Book*
- *J.E. Hendrix's Small-VM*
- *What FORTH Is?*
- *Several exciting Z80 utilities*
- *The Conference Tree*
- *PCNET information*
- *Several North Star tidbits*
- *An assortment of utilities*
- *And much, much more*


DR. DOBB'S JOURNAL
For Users of Small Computer Systems
A Reference Journal for Users of Home Computers
Volume Six
PCC The People's Computer Company Series

YES! □ Please send me the following Volumes of Dr. Dobb's Journal.
□ ALL 6 for ONLY $125, a savings of over 15%

Signature ______________________

Name ______________________

Address ______________________

City ______________ State ________ Zip ________

Card# ______________ Expiration Date ________

Please charge my:
□ Visa □ American Express
□ MasterCard I enclose □ Check/Money order

This offer expires March 31, 1984.

MAIL TO: Dr. Dobb's Journal P.O. Box E, Menlo Park, CA. 94026

These are reduced prices for Dr. Dobb's readers.

Vol. 1	_____	× $23.75	=	_______
Vol. 2	_____	× $23.75	=	_______
Vol. 3	_____	× $23.75	=	_______
Vol. 4	_____	× $23.75	=	_______
Vol. 5	_____	× $23.75	=	_______
Vol. 6	_____	× $27.75	=	_______
All 6	_____	× $125	=	_______
Subtotal				_______
Postage and Handling				_______

Must be included with order.
Please add $1.25 per book in U.S.
($2.00 each outside U.S.)

L7 **TOTAL** _______

Circle no. 66 on reader service card.

design team's familiarity with it. The S-100 card modem would operate as a fairly independent unit so that all timing and interface circuitry would be contained on the board.

Since most S-100 systems are based on the 8080 or Z80 family of microprocessors, the input and output for the modem were ported rather than memory mapped. The current design provides for 32 possible locations in the port addresses. S-100 standards provide for optional extended addressing for 16-bit port addressing, but this was not incorporated into the current design since 16-bit processors are currently more the exception than the rule. This will probably be changed as 16-bit processors become more common.

After the production of the basic specifications of the product, the first prototype was produced. This prototype incorporated the core modem and a wire-wrapped board containing the circuitry for the serial interface, speaker, and telephone lines.

To test a prototype and debug a wire-wrapped board is good for the soul. If you haven't done it, you're missing a lot of fun. If you have, you know what goes on. The first step in the process is to turn off the computer and take out every card on the bus. Usually an extender card is placed in a slot and the wire-wrapped board placed in another slot. The prototype card is then placed in the extender card. Carefully, you turn the computer on and keep one hand on the switch. As soon as the power is turned on, you look for smoke. If the prototype is "smoking," you quickly turn off the computer. This may sound funny, but ask anyone who has tested prototypes and they can tell you about at least one new board that smoked.

No smoke is the best thing the designer can hope for the first time a new board is powered up. If there is no smoke, the board should sit in the system for about one hour or so with the power on. This provides a chance for any component which may have a problem to burn in a bit. After about an hour, each component is checked for heat. Usually a finger on the part is enough to tell if there is a problem. Then a volt meter and an oscilloscope are applied to various circuits to check the voltage levels and pulses. Wire wrapping can result in bad connections that are tough to find.

The S-100 modem only had two wire-wrapping problems the first time it was tried. The real problem with wire wrapping is the transient problem: sometimes the board will work and sometimes it won't. Trying to track that problem can be a hair-pulling experience.

After the smoke test, the system is powered down and the other cards reinserted. The system is turned back on and the new card tried for the first time. With the S-100 card, it worked the first time. Data sent to the serial interface went to the modem and it worked. Several days of testing did show some minor flaws in the wiring, but they were easily corrected.

The next stage in the development of a new board is to have real circuit boards made. This requires developing a mask for photo etching and having cards actually printed. The S-100 card modem is a fairly simple design which requires only two layers (front and back of the card). The actual layout of the circuitry by a layout artist required about a week, and then about another week was required for the cards to be produced. The first real test of the product was in the printed circuit card.

After the cards are returned, a visual inspection is made, the components are soldered into place, and the test begins again. The smoke test is repeated. If the modem passes this test, then the full testing begins. Again, the S-100 modem required only minor changes. Two of the printed circuit trails were too close to each other, and one trail was on the wrong side of the board.

Correcting the problems was a simple task, but to insure a quality product one more round of printed circuit prototypes was developed and tested. These passed the test and production of the product began.

The chief designer of the S-100 modem is a good engineer who hates to write. So the hardest part of the development for him was the manual. Each designer is required to write the preliminary version of the manual which then goes through several revisions by the technical writing staff and the customer service staff before a version is sent to customers. After about six months, the version provided to customers is thoroughly reviewed and revised based on customer service feedback.

Design of the S-100 Card Modem

The photograph in Figure 1 (page 74) is a picture of the U.S. Robotics S-100 card modem. The S-100 modem may be divided into two functional parts: the modem and the serial interface. The modem is the functional equivalent of the U.S. Robotics Autodial 212A or Password modems. It is a fully functional autodial/autoanswer modem which operates at 0 to 300 baud and 1200 baud, according to the 212A protocol. (See below for more information about baud rates.) Instead of the RS232 connection between the computer and the modem, the S-100 communicates with the computer through the S-100 bus. Communication with the modem takes place through the commonly used I/O channels of the 8080, Z80, and 8086 family of microprocessors. Serial communication is through a 8251 USART.

The Modem

The S-100 card modem is an intelligent modem with autodialing and autoanswering capabilities. The S-100 modem accepts commands over the serial interface in the form of ASCII text characters, numbers, and special characters. An example of a command is:

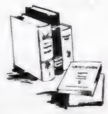
```
ATDT 555-8989
```

This command would autodial the telephone number 555-8989. Commands may be given to the modem for the following functions:

Enter Answer Mode
Dial
Echo back characters
Half/Full Duplex
On/Off Speaker
On/Off Result code
Set Number of Rings Before Answer
Change Disconnect Command Characters
Set Wait for Carrier time
Set Terse/Verbose Response
Use Extended Result codes
Reset

Once a communications link has been established, all information sent to the modem is passed over the communication lines. The only command the modem will accept is the command to disconnect (+++). The +++ sequence must be preceded and followed by one second of no data transmission. The character used as a three-letter sequence for disconnect can be changed by the user.

The S-100 modem will also automatically switch between originate and answer modes. When answering the telephone, it will automatically select between high-speed communication and low speed. Codes are sent to the terminal device



Changing Your Address?

Staple your label here.

To change your address, attach your address label from the cover of the magazine to this coupon and indicate your new address below.

Name

Address

Address Apt. #

City State

Zip

Mail to: DDJ, P.O. Box E, Menlo Park, CA 94025 indicating whether a high-speed or low-speed connection was established. The microcomputer can then adjust its own baud rate accordingly.

The S-100 modem also contains a speaker (upper left in the photograph in Figure 1) which will indicate the sound of the dialing tones and provide information about the status of the connection as it progresses. Once communication has been established, the speaker will automatically turn off. The user has the option to turn off the speaker by issuing a command. Although the speaker is contained within the case of the computer on the S-100 modem, it is generally loud enough to be heard through the computer case.

Baud Rates

The 212A communications protocol for modems provides for the transmission of data at 0 to 300 and 1200 baud. 600 baud may be found among Remote CP/M systems and other bulletin boards. The S-100 card modem has been designed so that it can handle 600-baud communications.

Since the serial interface and modem are contained on one card with no external RS232 interface, some of the signals designed for use with RS232 may be used for other purposes. In this way, the Request to Send line, sometimes used as an RS232 signal, has been used in the S-100 modem to provide access to more baud rates. When programming the 8251, if the RTS line is set to low, the baud rate will be divided in half. In this way, 1200 baud may be switched to 600 baud and 300 baud to 150 baud.

It must be remembered that 600 baud is really a fast 300 baud and not a slow 1200 baud. When a telephone connection is made, the protocol's baud rate is determined by the sounds the two modems make. These sounds follow a set pattern known as a protocol. The 300-baud and 1200-baud protocols are very different. Once a connection has been established at 300 baud, it is possible to change the baud rate to 600 baud at both ends and continue communications. Both sides of the connection must operate at the same baud rate or else garbage information will be received.

The design team believed that the large number of RCPMs and the common set-ups for communications at 600 baud required the inclusion of 600-baud capabilities.

The design process for the S-100 modem from beginning to the time the first products were shipped required about ten weeks. One of the real joys of helping to develop a new product is to see it boxed up and shipped out the door. There are other advantages. The chief designer of the team has serial number 1o of the modem hanging on the wall of his office and takes pride in showing it to visitors. Serial number two of the modem is in the computer used to write this article and was used to send the copy to *Dr. Dobb's* for printing. DDJ

Reader Ballot
Vote for your favorite feature/article.
Circle Reader Service **No. 198.**

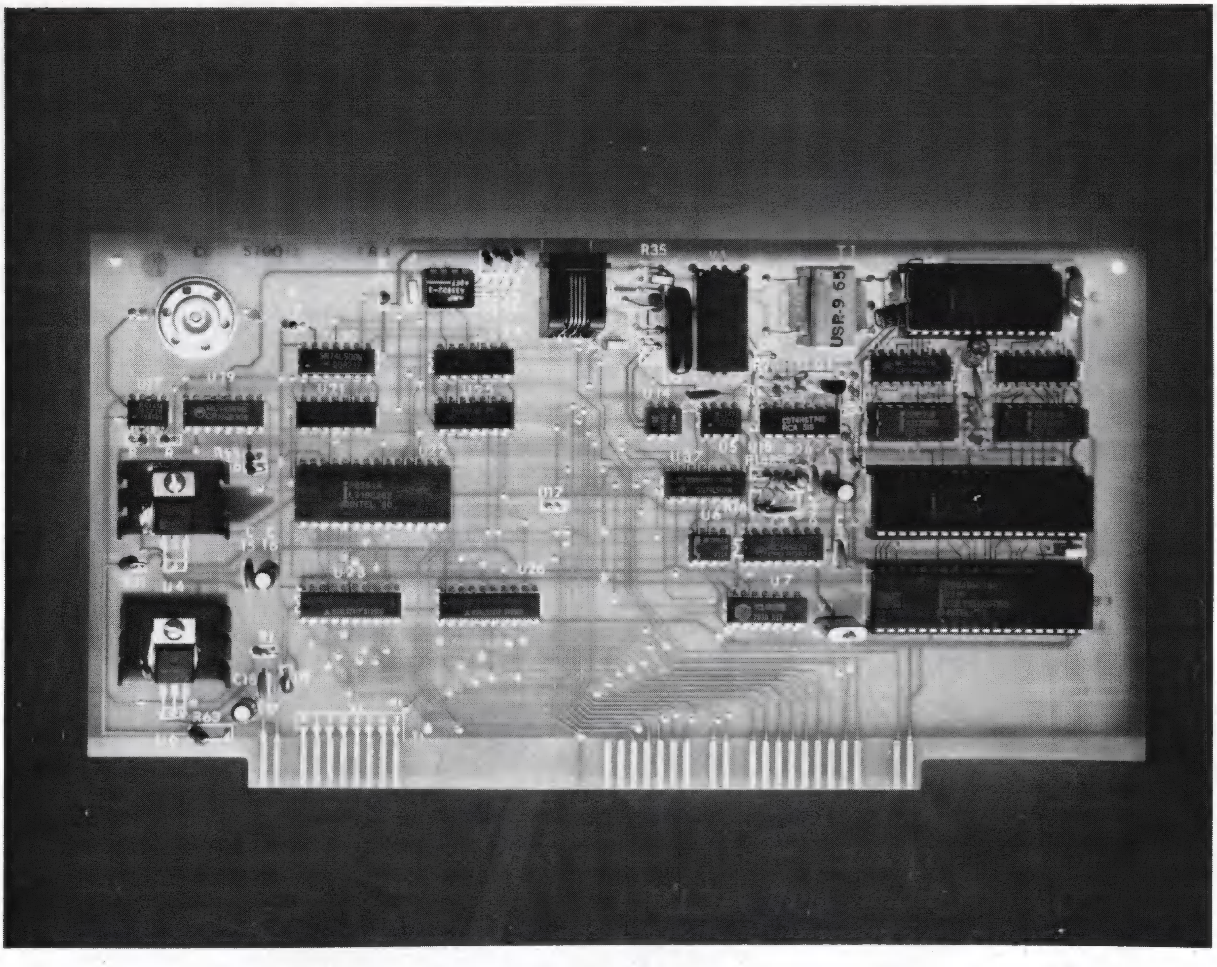


Figure 1.

GTEK INC.

DEVELOPMENT HARDWARE/SOFTWARE

HIGH PERFORMANCE/ COST RATIO

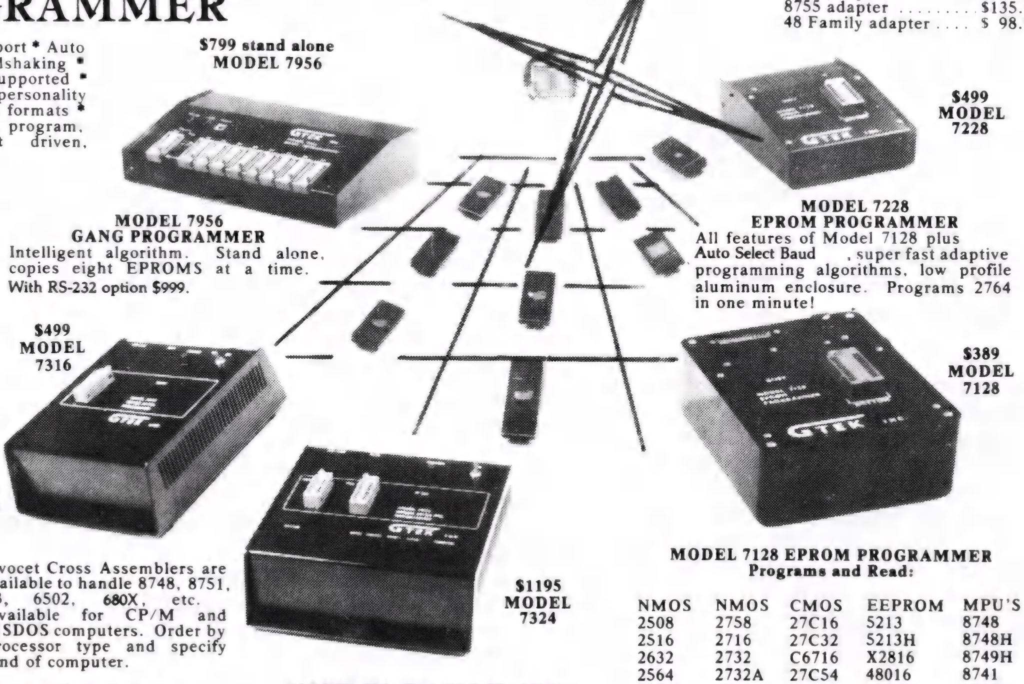
(601) 467-8048

EPROM PROGRAMMER

Model 7128-L1,L2,L2A	$209.00
Model 7128-24	$289.00
DR8 or DR5	$ 30.00
DR8PGX or DR5PGX	$ 75.00
Cross Assemblers	$200.00
XASM (for MSDOS)	$250.00
U/V Eraser DE-4	$ 78.00
RS232 Cables	$ 30.00
8751 adapter	$174.00
8755 adapter	$135.00
48 Family adapter	$ 98.00

Compatible w/all Rs 232 serial interface port * Auto select baud rate * With or without handshaking * Bidirectional Xon/Xoff and CTS/DTR supported * Read pin compatible ROMS * No personality modules * Intel, Motorola, MCS86, Hex formats * Split facility for 16 bit data paths * Read, program, formatted list commands * Interrupt driven, program and verify real time while sending data * Program single byte, block, or whole EPROM * Intelligent diagnostics discern bad and erasable EPROM * Verify erasure and compare commands * Busy light * Complete w/Textool zero insertion force socket and integral 120 VAC power (240 VAC/50Hz available)

DR Utility Package allows communication with 7128, 7228, and 7956 programmers from the CP/M command line. Source Code is provided. PGX utility package allows the same thing, but will also allow you to specify a range of addresses to send to the programmer, Verify, set the Eprom type.



$799 stand alone
MODEL 7956

MODEL 7956
GANG PROGRAMMER
Intelligent algorithm. Stand alone, copies eight EPROMS at a time. With RS-232 option $999.

$499
MODEL
7228

MODEL 7228
EPROM PROGRAMMER
All features of Model 7128 plus Auto Select Baud, super fast adaptive programming algorithms, low profile aluminum enclosure. Programs 2764 in one minute!

$499
MODEL
7316

$389
MODEL
7128

$1195
MODEL
7324

MODEL 7316 PAL PROGRAMMER
Programs all series 20 PALS. Software included for compiling PAL source codes.

Software Available for CPM,[1] ISIS,[2] TRSDOS,[3] MSDOS.[4]

1. TM of Digital Research Corp.
2. TM of Intel Corp.
3. TM of Tandy Corp.
4. TM of Microsoft.

Post Office Box 289
Waveland, Mississippi 39576
[601]-467-8048

Avocet Cross Assemblers are available to handle 8748, 8751, Z8, 6502, 680X, etc. Available for CP/M and MSDOS computers. Order by processor type and specify kind of computer.

Model DE-4 U/V Products hold 8, 28 pin parts. High quality professional construction.

MODEL 7324 PAL PROGRAMMER
Programs all series 20 & 24 PALS. Operates stand alone or via RS232.

MODEL 7128 EPROM PROGRAMMER
Programs and Read:

NMOS	NMOS	CMOS	EEPROM	MPU'S
2508	2758	27C16	5213	8748
2516	2716	27C32	5213H	8748H
2632	2732	C6716	X2816	8749H
2564	2732A	27C54	48016	8741
68766	2764		12816A	8742H
68764	27128			8741H
8755	27256			8751
5133				

Circle no. 31 on reader service card.

PolyForth™ II level 2 for your IBM-PC

■ A COMPLETE DEVELOPMENT SYSTEM $149

PolyFORTH II® is the latest version of the polyFORTH operating system. FORTH-Level 2 is a particular implementation designed to make available to the personal computer user the full power of FORTH®.

FORTH-Level 2 is a *complete*, fully integrated development environment. It includes the FORTH compiler, interpreter, full screen editor, assembler, virtual memory and multitasking capability.

FORTH-Level 2 takes full advantage of the hardware capability offered by IBM ranging from function keys to 8087 support.

■ MULTITASKING

User-definable task

A powerful vocabulary is provided to support multitasking. With the words BACKGROUND and BUILD, any number of background tasks can be easily defined. They are controlled with the words ACTIVATE and STOP. Memory availabililty is the only limitation to the number of tasks that can be created.

Printer and color monitor support

A printer task has already been defined. Words such as LIST, SHOW, PAGE . . . allow full control of the printer. Moreover, the color monitor can be used concurrently with the monochrome display defined as OPERATOR.

Efficiency

The multitasking algorithm was designed to maximize service to all tasks and is extremely fast. In fact, the round-robin polling cycle takes exactly one machine instruction per task. Similarly, the process of activating and de-activating tasks is very simple. Task activation requires only 9 instructions, while de-activation is done in 5 instructions.

■ TURNKEY COMPILER™

The Turnkey Compiler is used to build a bootable image of the application program including the underlying FORTH OS on a disk. Only binary object is transferred to the turnkey application disk, but source can be added if desired.

You may want to use the Turnkey compiler for booting quickly and executing automatically a frozen, dedicated application.

If you want to prepare application software for resale, the Turnkey compiler is the ideal utility for automatic preparation of your disk.

■ OPTIONS

■ 8087 support	$129
■ Data base support (include report writer)	$100
■ Multi-user capability	$129
■ Auto Documentor	$ 75
■ Graphics	$100
■ Forth Matrix Language	$100

> . . it's an excellent implementation and shows the benefit of FORTH, Inc.'s 10 years of experience with quality Forth Systems. It includes a number of advanced features that put it significantly ahead of many other versions of Forth.
>
> Barry Crawford
> P C Age review - Vol. 2.8, p. 66

FORTH TECHNOLOGY

in search of simplicity

432 15th Street   Santa Monica, California 90402

(213) 372-8493

Circle no. 29 on reader service card.

SOFTWARE REVIEWS

Friday!

Company: Ashton-Tate, 10150 W. Jefferson Blvd., Culver City, CA 90230
Computer: CP/M 2.x, CP/M-86, and MS-DOS
Price: $295.00
Circle Reader Service No. 137
Reviewed by R. A. Langevin

Friday! by Ashton-Tate is the son or, if you prefer, the daughter of dBase II. It provides most of the features of dBase II without requiring any knowledge of programming on the part of the user. In some respects Friday! provides capabilities that go beyond those of dBase II, since it permits users to custom design multi-line reports, create form letters with inserts from a data file, and define and print mailing labels – features not available in dBase II.

Environments

Friday! is available for a variety of current processor and operating system environments. These include 8080, 8085, and Z80 processors with CP/M 2.0 or higher and 8086 and 8088 processors with MS-DOS or CP/M-86. Memory requirements depend on the operating system used. A minimum of 64K is required for CP/M-80 and 128K for MS-DOS and CP/M-86.

In a floppy disk environment, dual floppies are required with a minimum of 126K available storage space. Friday! can also be used in a hard disk environment. A terminal or monitor that supports at least 80 columns and direct cursor addressing is required. Any printer can be used as long as it supports line lengths of 80 characters or more.

Features

Friday! is designed to support record-oriented files. A record can consist of up to 32 fields and can be up to 999 characters long. There are no provisions for files that span two or more disks, so the maximum file size that can be accommodated is limited by the space available on a single data disk.

All interactions with Friday! are by means of an extensive set of nested menus: there is no integral programming language as with dBase II. As a result no programming knowledge or experience is required to use any of Friday!'s facilities. The menus support the design of a file; entry, updating, deletion, sorting, selection, viewing, and printing of records; creation and printing of standard and custom reports; preparation and printing of form letters that include data from a selected file; and specification and printing of mailing labels.

In addition to these impressive capabilities, Friday! data files are compatible with dBase II and Lotus 1-2-3 files and can be read or written to by WordStar. Since the format of Friday! files is well defined, users can access them using the facilities of BASIC, Pascal, C, and other programming languages, a capability that will not normally be required.

After a file is designed, a standard Friday! data entry screen can be used or a custom-designed data entry screen can be readily produced with a series of screen layout menus. This is a useful capability. The user not only can design screens that are esthetically more pleasing than the standard data entry screen, but more importantly can create a number of custom screens for a single file. Each screen can be designed to present only the fields desired, so that selected information in the file (for example, salary information) is hidden from the individual doing file updates.

Two extremely useful features are available when creating or updating files. The first, a "ditto" capability, premits the user to specify a field or fields that contain fixed information (for example, a date or a price). This fixed information is automatically carried forward to each record as it is entered or updated, greatly speeding up the process. The second feature permits any field except the first to be a calculated field. Formulas for calculation of individual fields can be up to 60 characters long and have available a maximum of 500 characters. More important, the values of calculated fields are saved in the data file being worked on!

Reports can be created in two forms and the formats saved on disk. The first and simplest, the Quick Report, enables the user to specify the fields to be listed; give them titles that may differ from their internal names; specify their sequence across a report line; and print the corresponding report using up to 15 relational rules to select the records to be used. Although Friday! always maintains the records in a file sorted on the first field of the record, temporary sorts can be defined on as many as five fields for the production of reports.

The second and more flexible report form, the Custom Report, permits the user to select the fields to be printed and to locate them anywhere on the page. This flexibility, not available in dBase II, affords the opportunity for considerable creativity in report layout. Since the Custom Report format can incorporate any text whatever in addition to data from the records selected, it can also be used to create simple, single-page form letters, as well as be adapted to the creation and printing of mailing labels.

Documentation

Friday!'s documentation is among the best I've ever seen. Its 207 pages are attractively printed and bound in a sturdy three-ring notebook. The back of the notebook is hinged so that it will stand up on a desk. All in all, very attractive packaging!

The documentation consists of a brief overview of Friday! and a series of six lessons that should be worked through at a terminal. These lessons will quickly teach anyone how to use all of Friday!'s features. The next section of the manual, almost 75 pages long, provides an exhaustive description of all the prompts that appear on the screen as Friday! is used. A glossary is also provided to help new users over the rough spots of terminology.

A series of eight appendices and a good index conclude the documentation. The appendices describe Friday!'s sorting order, file naming conventions, relationship to dBase II, field renaming procedures, use of backup files, working and backup copies of Friday!, installation instructions, and the use of Friday! with Lotus 1-2-3. After the index a handy reference folder summarizes Friday! commands and procedures. This is a documentation model that others could well follow.

Problems

Unfortunately, nothing is perfect. There are a few things in Friday! that I would like to see changed.

Although the user can create a custom screen for data entry or update and can place fields anywhere on the screen, the sequence that the cursor follows as it skips through fields is the sequence in which the fields were originally defined in the record layout. While this is not a real problem, it can be disconcerting to see the cursor jumping around the screen.

I consider the two built-in, unchangeable default limits to be a more serious problem. Specifically, it is impossible to print a line longer than 160 characters, and the physical length of a page is always taken to be 66 lines. Many printers are capable of printing lines substantially longer than 160 characters, and the extra width can be extremely useful. It's not clear why Ashton-Tate chose such a short line length as an upper limit. The built-in assumption of 66 lines per page can be lived with, but many times it is more convenient to print a report on paper that is 8½ inches high and 11 inches wide; this is difficult to do with Friday!.

Summary

The foregoing criticisms aside, Friday! is a powerful file handling package that affords most of the capabilities that a user would normally want. Best of all, these capabilities are now accessible to the nonprogrammer. Friday! should do well in the marketplace.

SAL/80

Company: Protools, 24225 Summerhill Avenue, Los Altos, CA 94022
Computer System: CP/M 2.2
Price: $59.95
Circle Reader Service No. 133
Reviewed by Jim Kronman

SAL/80 is a software package that adds standard "structured" control operators to the 8080/85 assembly language and can also, at the user's option, produce some code optimized with Z80 instructions. SAL/80 is designed to be used with the Digital Research, Inc. MAC and RMAC macroassemblers. In addition to the control structures, built-in library functions are provided. Newberry Microsystems refers to SAL/80 as a "language," but it is actually a language enhancement, used in the same way as RATFOR is used with Fortran.

Structured programming is a lot like religion: either you believe or you don't. I will not try to sell you on the concept in this review, but if you believe in minimal use of GOTO, you will appreciate SAL/80.

This review applies to version 2.1 of SAL/80 running under CP/M 2.2. The author of SAL/80 is Steve Newberry and it is marketed by Steve's company, Protools. The current price for SAL/80 including a 200-plus page manual is $59.95. The price includes a free update to version 3.0. SAL/86 is or will be available for 16-bit processors.

What It Is

SAL/80 provides these familiar control structures:

```
DO/ENDDO
FOR/UNTIL/STEP/ENDFOR
IF/ENDIF/ELSE/ENDELSE
LOOP/UNTIL/ENDLOOP
REPEAT/ENDREPEAT
WHILE/ENDWHILE
SELECT/CASE . . .
    CASE/ENDSELECT
```

EXIT and GOTO are also provided.

The control structures are supported by a wide range of boolean expressions of the form X,rel,Y where X and Y (with a few limitations) can be any register, register pair, memory reference, or constant. The "rel" in these expressions can be either the usual comparisons (such as greater than, equal to, etc.) or one of several CPU status flag conditions (e.g., carry set, minus, etc.) that require no argument (X and Y are null and the PSW is implied). For convenience, IFCALL and IFGOTO are provided to allow a conditional call and goto consistent with the other operations.

The SAL/80 built-in functions include register save and restore functions. Most of the built-ins are designed to facilitate program-to-user interface via the console.

Several SAL/80 library files are also provided on the distribution disk. They include block move, string search and comparison procedures, and multiply and divide.

SAL/80 is designed to work in a 8080/85 environment but optionally will generate Z80 code for relative jumps and the "DJNZ" instruction in loops.

Using SAL/80

SAL/80 version 2.1 is implemented as a set of macros that must be invoked as "MACLIB SAL/80" at the beginning of your assembly language program. You include the SAL/80 instructions as part of your assembly language program code. When it is time to assemble the code, SAL80.LIB and any of the provided ".LIB" files you have referenced must be on the same disk as your source code. You run MAC or RMAC and the program is assembled.

SAL/80 Documentation

The SAL/80 manual has four chapters and three appendices. The chapters are indexed. The compiler source code has its own index, this source listing plus its index taking 72 pages.

Chapter One, "Overview of Essentials," explains the syntax, content, and use of SAL/80 in 45 pages. A generalized example of each construct or function is given in schematic form. For example:

```
LOOP?
    [ < loop body > ]
UNTIL?  < bool expr >
    [ < more loop body > ]
ENDLOOP?
```

This example points out one feature I have so far not mentioned: all SAL/80 instructions are terminated with "?".

Chapter Two, "Tutorial: Maintainable Programs," is a 31-page essay on software design using a real-life (if somewhat contrived) example. All of the steps from design concept to finished product are covered.

Chapter Three, "Tutorial Worked Example: MEMTEST," is the complete source code listing of the example used in the previous chapter. It includes its own index (for the program) and occupies 50 pages. MEMTEST is a memory test program.

Chapter Four is called "General Observations" and talks about using MAC and RMAC and the application of some of the features of SAL/80.

Appendix A of the manual is the complete SAL/80 source code. The distribution disk file SAL/80.LIB, however, does not look like the source code. It has been, in Newberry's words, "krunched" in order to save disk and TPA space. Appendix B lists the source code for the error trapping portion of SAL/80.LIB. Appendix C is a summary of SAL/80 commands.

Performance

SAL/80 does what it claims to do. It runs, has a few bugs, but is well supported by the author. I can only fault it for being *slow.* On the other hand, RATFOR is no burner with Fortran, either. Steve Newberry has recognized the speed problem with his MAC.LIB approach and is currently preparing a preprocessor version of SAL/80 written in the C language. Purchasers of version 2.1 will receive version 3.0 free when it is available.

The code produced by SAL/80 is reasonably close to being as compact as the code you would produce if you coded in "pure" assembler. For example, a small menu-driven program I wrote with SAL/80 to send set-up commands to a printer was 1.4K bytes, and a similar program for another printer from "pure" assembly code was just under 1.0K bytes. Although some minor inefficiencies have been created in exchange for generality, SAL/80 generates more compact code than any level language compiler I have used. I have not noticed any case where execution speed has suffered. My experience with SAL/80 has been with business applications; if you are coding a real-time control application, however, you might encounter critical areas where fine tuning is required. The same would be true if you were coding in "pure" assembler.

Most of the constructs work as you would intuitively expect them to from experience with higher level structured languages such as Pascal or C. I found the SELECT construct, however, more limited than its CASE counterpart in Pascal. SELECT/CASE. . .CASE/ENDSELECT is actually more like Fortran's computed GOTO statement. The first choice (CASE) *must* be 0, then 1, and so on. This is stated in the documentation, but I had a hard time understanding what it meant, probably because I wanted it to work like Pascal! Another enhancement that would have been nice is if the X,rel,Y operations (relational operators, *rel*) would accept the syntax "A" to represent 41 hex and so on (like CHR(A) in Pascal or the literals in standard Intel assembler code).

I have not succeeded in getting the example program, MEMTEST, to run on my computer. I must admit that I have not given much priority to this and therefore have not spent much time trying. I have had no substantial difficulty, however, getting any of my own applications written with SAL/80 running.

The only other operational criticism I can make is that the error messages are not always sufficiently clear to allow debugging without resorting to examination of a ".PRN" file with the macro expansions included. This might not seem like a big deal until the first time you look at such a file from SAL/80. Some of those innocent-looking constructs and built-ins can generate gobs of code!

I have found no problems in the built-in and library functions supplied with SAL/80, but in some respects they are a reinvention of the wheel. My personal preference is for a set of routines like Richard Conn's SYSLIB functions, which are in the public domain. I find some of the SAL/80 procedure names (such as "compare$strings:") to be longer than necessary; I type a lot, but that doesn't mean I like to do it.

Evaluation of the Documentation

When I originally ordered SAL/80, I was motivated to do so by the documentation more than by the program itself. I thought it would be an economical way to learn more about MAC and RMAC and to pick up a few new programming tricks. My aims were accomplished, but I feel the documentation could be improved in some areas.

The manual is better than many. It *is* thoroughly indexed. In fact, I wish more software publishers would be as thorough as Steve Newberry has been in indexing his manual.

The greatest improvement in the manual would be to provide a real, concrete, illustrative example with the definition of each command. This would go a long way toward making SAL/80 immediately useful. I found I had to try each construct a few ways until I got clear in my mind what each function could and could not do. Some real examples accompanying each function description would have saved me hours of time.

Another welcome addition would be a simple, columnar quick-reference list of all reserved words and a compact quick-reference guide for the commands. I would like to see the commands, built-ins, and so on defined one to a page in the manual for ease of reference.

Support

I have nothing but praise for Newberry's support of his product. I am somewhat reluctant to print constructive criticism about SAL/80 because he has been so responsive to past inputs (from me and others) I am reasonably sure that most of my criticisms will have been answered by changes in the documentation by the time you read this review. His initial release of version 2.1 had some bugs in it, but it was released at a discount. In addition, anyone who reports a real bug receives a free update to version 3.1 for the effort (version 3.0 is already included when you buy 2.1). Corrections fixing the identified bugs have been mailed to users in a timely manner.

Conclusions

If you must do assembly language programming and want to generate more readable and maintainable code, then you should investigate Protools' SAL/80. It has the potential to give you some of the advantages of the higher level languages while retaining the control and efficiency of machine-level programming.

FILEBASE

Company: EWDP Software, P. O. Box 40283, Indianapolis, IN 40283
Computer: 64K Z80 CP/M
Price: $75.00
Circle Reader Service No. 135
Reviewed by Dennis Cashton

The object of many user-written programs, as well as many purchased software packages, is the manipulation of data in one form or another. Quite often what happens to data is not exactly what you had in mind, and you wind up having to write some sort of fix-up program to massage the data. Or you have a data entry package, a prime example of one application that cannot stand alone, and want to write a program to handle the data after entering them. On the way to your accounting programs or customer file processing programs, you are bound to run into a need to sort, extract, globally change, or otherwise shift things around in a file. In all these cases, it would be nice to have a listing of the data as they came from your source file or after they are sorted. Well, it seems that the folks at EWDP Software have a pretty good answer to your problems.

FILEBASE is an effective utility for performing many useful file handling functions. With its clean, no-frills, menu approach, the user can select to:

1. SORT a file (no exclude or include)
2. SELECT/EXCLUDE only (no sort)
3. SORT with SELECT/EXCLUDE
4. LIST record positions and fields (to screen or printer)
5. ADD new fields to an existing file
6. CREATE a new file
7. APPEND data to an existing file
8. WRITE a subset of fields to a new file

In going through the manual for FILEBASE, the first thing I noticed was that the people at EWDP have gone to a lot of trouble to make the documentation and the program very easy to use. They assume no prior knowledge on the part of the user, and you are led by the nose through every function. Examples are provided for each function described, and explanations for every example.

It would be possible to load the program and use it without reading the documentation, but some of the features that make FILEBASE such a versatile tool would not be obvious from the menus provided. One example of such a feature is the ability to sort on a last name within a name field by using the "L" suffix on a field number. You have the same capability to get at zip codes by specifying a "Z" suffix.

It's little extras like this that make FILEBASE nice to use and especially suited for mailing list applications. In fact, the nice people at EWDP Software sent a draft of a brochure describing FILEBASE in such an application. Especially nice is the feature that allows you to set a pause between each record on printing, as well as the number of lines to skip between records and the tab position for start of printing. This makes it very simple to print envelopes. Other print controls make it easy to handle any size or shape of label, card, envelope, or paper.

Other niceties include the ability to merge input files and to create multiple output files from one or two inputs. Comparators for SELECT/EXCLUDE can be:

EQ – equal to
GT – greater than
GE – greater than or equal to
LT – less than
LE – less than or equal to
(now the good stuff)
BT – between values
LS – one of a list of values
RR – range of record numbers

Sort can be ascending or descending, and the number of fields that can be printed across the page depends on the specified printer paper width.

There are, as always, several cautions and catches to watch out for. First, the documentation I received was stamped "OUTDATED MANUAL – SAMPLE ONLY" and was for version 1.1. The diskette supplied was labeled Release 2, and an attached note stated that the Release 2 manual would follow on publication. I can only hope the functional differences are not significant.

The manual refers often to the use of "comma delimited fields," but this is standard for many data entry programs and most versions of BASIC. The use of files "external" to FILEBASE is also stressed. All this means is that your input files will not be manipulated or destroyed by inadvertent bungling. In my many years of experience with inadvertent bungling, I consider this a definite plus.

Unlike the common data base management systems, FILEBASE does not need to have fixed-field, structured records. It will actually read the file to find out the field types and lengths.

FILEBASE has its limitations, but it is still more than sufficient for most applications. The limits are:

Record length – 4096 bytes max
Fields per record – 40 max
Input files per execution – two
Output files per execution – two
Records for SELECT/EXCLUDE – 15000 max
Records for sorting – 4000 max
File size – 240K max (more possible, but requires extra processing time)
Numeric sort precision – 10 digits

One interesting note: The manual says it needs 64K to execute, but it ran with no noticeable degradation on my 52K CP/M system.

I find it difficult to express how pleased I was from the moment I put the disk in and typed in FILEBASE to get it started. Everything worked as stated, if not better, and I instantly thought of a dozen things it would come in handy for. When you weigh utility against price, I don't see how a person who deals with any kind of data files could do without this program. It is everything you could ask for, in documentation, function, and price.

EWDP has informed us that Release 3 will be introduced at SOFTCON in New Orleans, Feb. 21-23. Enhancements include random access by user controlled indexing of any field and ability to select or exclude records by sub string searches. – Ed.

Oubliette

Company: Centaur Software, 501 Jackson, Charleston, IL 61920
Computer System: CP/M
Price: $39.95
Circle Reader Service No. 141
Reviewed by W. James Wiggins

Venturing down a dark passageway, you come to a doorway. Peering cautiously in, you see before you a large bronze dragon; he suddenly spies you and . . . the battle is on.

This could easily be the scene in the adventure game called Oubliette. According to Centaur Software, Oubliette is French for dungeon or maze. The game follows the type of game made popular by TSR, the Dungeons and Dragons series – and follows it very well.

This author has been addicted to adventure games for about two and a half years now and has played "several," those with graphics and those without. When Oubliette was given to me to review, I therefore felt confident that the game would be a snap to play, having had so much experience before with "games of this sort." I therefore sat down to play it without any more than a cursory stare at the artwork on the front of the very nice looking manual (after all, who reads manuals or instructions?). After wiping out my entire party (fortunately on my backup disk) I decided it might be good to at least look up some of the magic spells. Two hours later, I finally got back to the game . . . not because the manual was that hard to understand – it was that engrossing.

Centaur documents not only why the world of Oubliette is the way it is, but they even translate the magical spells so that we lesser mortals can understand them. The manual flows well – there are charts and graphs clearly and succinctly comparing various races and classes; there are in-depth explanations on magical and clerical spells; there is even an almost complete map of the first level of the dungeon.

The monsters, races, and classes will seem familiar if you have played Dungeons and Dragons before; the thrill of being able to run up to six characters by yourself and have the Dungeon Master be totally impartial will hook you on this game, though.

The only drawback to the manual is the documentation of how to set the game up for your terminal. The disk includes a program to help you configure Oubliette for your terminal, and unless you have a strange terminal like I do (I was playing it on an Apple II with a Videx 80-column card), you should be able to set it up for your terminal very easily.

Now that I am an expert at the game, let me tell you the best way to play it. First, you need to build up your party and then . . . well, maybe I'll let you figure it out (maybe I will too). The one thing that I will tell you is that when I called and spoke to the people at Centaur, they asked how far down in the dungeon I had traveled. I admitted to going down to only the third level. They said that most people who went down farther seldom lived to tell about it. I wonder what's down there?

In case I haven't given you a clue, the game is very good – realistic, engrossing, and well thought out. I would say that it is the most engrossing game that I have played in a long time. ■■J

BOOK REVIEWS

C Programming Guide

by Dr. Jack Purdum
Published by Que Corporation
$17.95, 250 pages
Reviewed by Dr. Joseph B. Rothstein

Just a few years ago, programming a microcomputer in a high-level language meant one thing: BASIC. Now that programmers can take advantage of microcomputer implementations of Pascal, Fortran, Lisp, APL, Forth, Ada, Cobol, and others too many to name, it appears likely that no language will again have the dominant position BASIC has enjoyed.

But from this Babel of programming languages, a recent entry is making a strong bid for serious consideration in the coming years. That language is C, a product of Bell Laboratories, the research arm of AT&T.

Though C has developed a vocal and growing following among minicomputer users – particularly those working in the Unix operating system environment – it has only recently become available on microcomputers.

The specification for C was published in *The C Programming Language* by Kernighan and Ritchie (published by Prentice-Hall, 228 pages). Any serious C programmer will eventually want to study it, but it includes only one chapter of tutorial introduction. Its terse, somewhat scholarly style is probably not appropriate for many microcomputer users, who have widely varying backgrounds and might want more introductory material.

C Programming Guide addresses this new group of potential C programmers. Purdum is the author of an implementation of C for systems using the Z80 CPU. Though the *Guide* could serve as an introduction to any version of standard C, its style and approach are similar to other popular microcomputer programming tutorials.

In nine chapters, Purdum covers the syntax and use of C in an orderly, conversational manner using numerous example programs and program fragments. Because of its extensive set of operators and data types, learning C can be a challenge to those who have only used BASIC or Fortran. But Purdum's incremental approach minimizes the difficulty, stressing hands-on experience of C and occasionally offering side-by-side comparisons of similar programs in C and BASIC.

Most of the programs included in the text should run on any C compiler that is Unix version-7 compatible, so the reader with access to such a compiler can get immediate feedback by entering and testing the programs.

Sometimes the text raises questions that can only be answered by running a particular program. The student is expected to enter, compile, and run the program and observe its results. This approach may limit the usefulness of the book for those without access to a compiler. The preface mentions "two underlying assumptions: (1) the only way to learn a language is to write programs with it, and (2) learning is made easier if you can visualize what a statement does," so perhaps that was the author's intention.

While Purdum often suggests "playing around" with a particular statement or function, specific exercises (and solutions to selected exercises) would also be helpful. The student would then have the option of choosing the intuitive, exploratory approach or a more orderly, disciplined one.

Despite these concerns, the *Guide* is a worthwhile tutorial introduction to a powerful and versatile language. C has been called a "mid-level" language, offering the direct control over machine architecture which we normally associate with assembly language, while including a range of constructs and operations typical of high-level languages. Execution speed is comparable to assembler, with the important bonus that C was designed to be portable. Unlike assembly language programs, a C source file can be recompiled to run on a variety of CPUs, generally with little or no change required. These features help make C an attractive language for systems programming – a fact that has not been lost on some of the biggest software houses, who are reportedly writing more and more of their programming languages and utilities in C.

In general, C programs consist of a series of functions, any of which may be placed into libraries and used again in other programs. The capability to develop and test a function only once, then use it freely without the need to modify or even test it, is one of the most attractive aspects of C. The text thoroughly and clearly explains the nature and use of functions, and how they are combined into complete programs.

The index is thorough and easy to use, and appendices contain a syntax summary of C and a list of commercially available C compilers for microcomputers. Along with publishers' names, addresses, and retail prices, Purdum offers brief comments including the degree of compliance with the Kernighan and Ritchie standard. To his credit, Purdum avoids making his book an advertisement for Eco-C – his own version of the compiler. In fact, his association with that particular C compiler is not explicitly mentioned, and a comment describes Eco-C only as "a very nice implementation."

For all its importance to minicomputer users, there are surprisingly few tutorial introductions to programming in C. Though *C Programming Guide* is no substitute for Kernighan and Ritchie's text (nor was it intended to be), it may serve to introduce the microcomputer user to a powerful programming language, one likely to assume an increasingly important position in the microcomputer world.

Microcomputer Graphics Techniques and Applications

By Donald Hearn and Pauline Baker
Published by Prentice-Hall, Inc.
$24.94 cloth or $19.95 paper, 302 pages
Reviewed by Robert Ashworth

This textbook is divided into fourteen major chapters dealing with almost everything a person would want to know about computer graphics on a small computer. Sixteen color figures showing the latest shading techniques as well as many black and white illustrations make this an informative and enjoyable book to read, study, and even give to a friend for browsing.

Hearn and Baker divide the book into five sections. The first four present an overview, the fundamentals of microcomputer graphics, and details on special effects and 3D. The fifth section contains program design, special techniques, simulations, computer-assisted instruction, budget nutrition charts, and game playing. The 92 BASIC programs progress from a simple nine-line picture of of a fish to a 180-line, three-page program. Each chapter concludes with several challenging projects for further programming.

The approach is straightforward, logical, and informative. The graphics command conversion table in the appendix is very useful; it gives the equivalent graphics commands for the Apple, Radio Shack, and the IBM PC. This is an enjoyable book to read and study, and it offers many varied illustrations to excite the eye and challenge the mind.

Introduction to Numerical Computation in Pascal

by P. M. Dew and K. R. James
Published by Springer-Verlag, New York, Inc.
$16.00, 291 pages
Reviewed by Robert Ashworth

The first part of this book focuses on Pascal with an emphasis on the fundamental techniques needed to exploit the modular structure of the language. The authors have developed very readable code with particularly clear descriptions to show the results of quality mathematical software.

The central theme is a description of "mathlib" which is a collection of routines available from the Editor of the Computer Science series of the publisher. These run under UCSD Pascal on the Apple II microcomputer. A ten-page summary of all the routines is given in the appendix.

The authors take an algorithm or procedure for the solution of a given problem and then show, step-by-step, how to make it more comprehensive, accurate, or efficient. Stress is placed on: stability, i.e., the algorithm is not sensitive to rounding errors; accuracy, i.e., the algorithm performs according to the documentation; efficiency, i.e., the algorithm is a function of time and memory required; robustness, i.e., the algorithm solves correctly the problem it was intended for; adaptability, i.e., the ability to move a program from one computer to another with a minimum of modification; error control, i.e., the need to have results correct to a specified number of decimal places; and the respective mathematical method of analysis. Part I concludes with a detailed discussion of errors and how to manage them on your computer.

Part II explores the development of mathematical software. Different methods are given for the solution of nonlinear equations in one variable. The study of systems of linear equations is followed by different methods and an analysis of the strengths of each approach. Finally, the last two chapters deal with the fixed-point rules and the adaptive methods for numerical integration.

The text meets the authors' goal of covering core material in numerical analysis and methods for producing reliable mathematical software. Solutions are given for all exercises and four helpful appendices complement the well-designed text. It should be given a high priority for Pascal programmers.

Automation of Reasoning

Classical papers on computational logic, Vol. 1, 1957-1966; Vol. 2, 1967-1970
Edited by Jorg Siekmann and Graham Wilson
Published by Springer-Verlag: Berlin, Heidelberg, New York, 1983
$35.00 Vol. 1: 525 pages, Vol. 2: 637 pages.
Reviewed by Jay Halcomb

These are the first two volumes of a planned series devoted to automated theorem proving. The volumes collectively contain 61 articles discussing mechanical theorem proving, which has applications in automated program verification, program synthesis, and deductive data bases. The papers are of the highest quality, selected by an international committee of researchers in the field, and include translations of previously untranslated literature.

On the whole the papers are focused in scope, approaching computational logic in its strictest sense: the application of theories of first-order deduction and semantic methods to proof of mathematical or logical theorems. A few of the papers allude to possible wider applications, such as generalized problem solving and program verification, but the core concern is with theorem proving in logic and mathematics. In this sense the title and cover blurb are slightly misleading. However, for readers with professional or avocational interests in this field, the book is an excellent unified reference source.

The volumes contain three historical articles especially useful to the new student or casual onlooker:

Wos and Henkin: "Automated thoerem proving, 1965-70"

Davis: "The prehistory and early history of automated deduction"

Maslov, Mints, and Orevkov: "Mechanical proof search and the theory of logical deduction in the USSR"

Background

The philosopher G. Leibnitz was perhaps the first to seriously envisage the possibility of algorithmizing and even mechanizing human reasoning. He proposed a *calculus ratiocinator* and *lingua characteristica* "to bring under mathematical laws human reasoning, which is the most excellent and useful thing we have . . . the mind will be freed from having to think directly of things themselves, and yet everything will turn out correctly." But this vision lacked foundations until the growth of mathematical logic through the work of Boole, Frege, Russell, and Whitehead and the creation of recursion theory by Turing, Church, and Kleene.

The subsequent implementation of these mathematico-logical theories in actual machines in this century brought the matter to a practical head. But until very lately enthusiasm and optimistic speculation have been more characteristic of the field than have actual results. However, the recent announcement of a proof by computer of the famous four-color problem, a proof that apparently was not to be achieved in any other manner, established a substantial and important mathematical theorem, and the automation of human reasoning began to come of age.

Work on theorem proving in recent decades first concentrated, naturally perhaps, on the verification of known results, testing theories of problem solving rather than attempting new results. Thus in the late 50s the logician Hao Wang was able to deduce all of the theorems of Russell and Whitehead's *Principia Mathematica* in a few minutes on an IBM computer. Later workers began attempting to formulate new results in the areas of group theory and model theory and achieved some noteworthy successes, establishing previously unknown theorems of some significance. But the crowning feat to date remains the aforementioned mechanical solution of the famous four-color problem.

Briefly, the four-color problem questions whether four colors will always suffice for coloring any map on a two-dimensional surface. Much work was expended on this deceptively simple problem during the past century, but the general case remained elusive until the announcement in 1976 of a computer-assisted solution by Appel and Haken ("Every planar map is 4-colorable," *Bull Am Math Soc,* No. 82, 1976). It should be noted, however, that this success relied on a prior ingenious reduction of the problem to a large but finite number of cases, which the machine searched as the "proof," so the result is not unalloyed. This work is not discussed in the present volumes, due to its more recent date, but will doubtless appear in the next of the series.

Schools of Automated Reasoning

There are two general approaches to the simulation of reasoning by computers. One is the implementation of strictly deductive theory for a specific class of problems (such methods achieved the results mentioned above). The other is a heuristic, generalized, problem solving approach that requires the computer to generate "reasonable" search directions in problem solving, in this way emulating the human mind. This approach is associated with the work of Newell, Simon, and Shaw, early pioneers in artificial intelligence (AI).

Lately we have come to see some amalgamation of these techniques. Although the availability of cheap memory and higher speeds, along with a sophistication of inference rules, has greatly fortified the deductive approach, a judicious use of heuristic rules clearly is also necessary to allow the computer to escape from an infinite drudgery of trivial computation. In 1961 Marvin Minsky, one of the AI leaders, wrote: " . . . it seems clear that a program to solve real mathematical problems will have to combine the mathematical sophistication of Wang with the heuristic sophistication of Newell, Shaw, and Simon."

Summary

The present volumes concentrate on the deductive approach, perhaps justifiably since these techniques have so far yielded more substantial results. While Eliza-like programs are striking, they are not very sophisticated, either logically or heuristically.

In summary these are not books for the general hardware hacker but are of a rather narrow academic interest. They should be welcomed by their intended audience.

■■J

16-BIT SOFTWARE TOOLBOX

by Ray Duncan

PC-DOS Close Function

I was tipped off to the first topic for this month's column by an old COBOL programmer who had been trained to close every file in sight if execution of his program was terminated due to some error condition. When he tried to persist in this habit under PC-DOS, his files began to mysteriously disappear. They would still appear on a directory listing, but the file lengths were displayed as zero bytes and the CHKDSK utility was finding all sorts of "lost clusters" that it had to clean up.

The problem turned out to be that PC-DOS is not checking to see if a file control block has been properly activated by a previous successful OPEN or MAKE call before carrying out the CLOSE operation. Apparently it just looks up the corresponding entry in the disk directory and copies in the zeroed-out file control block information, effectively destroying it. PC-DOS then gives the calling program back a return code indicating success (AL=zero). This is a really surprising bug, since there are several ways that the operating system can quickly assess whether a file control block is valid or not.

I have vivid memories of the reception given to the poor soul who originally reported the IBM PC BASIC math routine errors, and also of the rather indignant letter previously directed to this column by the grandfather of PC-DOS 1.1 (Tim Paterson). Therefore, I am supplying a detailed example program in Listing One (page 85) that can be used to verify that this problem exists in both PC-DOS 1.1 and 2.0. I haven't had time yet to explore what happens when you ask for other disk operations (such as read or write record) with an unopened file control block, but I'm sure it will turn out to be most amusing.

The Microsoft Assembler

The Macro Assembler for the IBM PC, which was supplied by Microsoft, has come in for a little criticism from time to time because of its lack of speed and its appetite for memory, but it has escaped otherwise unscathed in all the magazines I read. Its performance and size can be attributed to the fact that it was written in Pascal (a more unlikely high level langauge in which to write such a systems tool can hardly be imagined), and in spite of its handicaps it has been in the Softalk Top 30 month after month.

Certain "features" of the Assembler make me conclude that it was written to a set of specifications by a hotshot bunch of Pascal hackers who never even tested it, let alone used it for any kind of job in the real world. The first evidence for this is found in some bizarre listing notations. Although every assembler I have encountered will show you the real value of an equate in the object code column of the listing, the Microsoft Assembler gives you a signed hexadecimal integer (see Listing Two on page 87, line 15).

The authors of the Assembler also chose to ignore the byte-swapped nature of 16-bit values on Intel processors in their listing format. In line 21 of Listing Two, the byte at address 0000 appears to contain 00 while the byte at 0001 contains 01 – of course the reverse is what is truly found in the object code. Patching a program working from such a listing is unnecessarily confusing.

But as long as we're ragging on the boys at Microsoft, how about pointing out some really gross bugs.

The SHL and SHR operators are not mentioned at all in the IBM version of the Assembler manual, except in a brief example on page 4-21 which states that 101B SHL (2*2) should yield 01010000B, or 50H. From this you can infer that the SHL operator is used in the form "data SHL shift_count," which in fact agrees with the Intel specification. Unfortunately, no one ever checked the actual operation of the Assembler against the example in the manual. By experiment, we find that the Microsoft Assembler expects the shift count first (see lines 29-37 of Listing Two), in conflict with both the Intel and Digital Research 8086 assemblers.

The SHR operator, on the other hand, behaves entirely unpredictably. In some cases it seems to act like the SHL operator, in others it doesn't return any understandable result at all (see lines 39-53).

Intel defines SHR and SHL as performing "logical shifts," bringing zero bits into the vacated positions. The Microsoft Assembler is inconsistent on this point (see lines 57-62). My guess is that the authors are ANDing the shift count with a mask of 0FH, instead of testing for a shift count greater than 15 which should return a zero result for any data.

The logical operators EQ, NE, GT, GE, LT, and LE all perform erroneously as often as not (see lines 64-91). I surmise here that the operators are somehow ignoring the signs of decimal integers. Other than that, the operators apparently treat their arguments as *unsigned* 16-bit integers, a fact that is not documented in the manual but can be deduced from comparing hex values (for example, 07FFFH LT 08000H returns TRUE).

The Not operator works for positive decimal integers and all hexadecimal integers, but fails with strange results for negative decimal integers (see line 95). Similarly, the logical Inclusive Or operator gives incorrect results with negative decimal integers.

Lastly, the Exclusive Or operator appears to work as an Inclusive Or instead (see lines 107-110). Don't imagine for a moment that we have covered all the bugs! These were located in just a few minutes' work, and I have been tipped off to others that should be good for several more *DDJ* columns.

The moral of this month's story seems to be: Avoid use of exotic or complex expressions, and perform any calculations involving logical operations at runtime rather than assembly time. I had to laugh when I found the note on page D-14 of the Assembler manual that said, "In general, be careful when patching in hex." Usually, this would be advice too obvious and banal to be worth the ink, as though someone were trying to say, "In general, be careful when jumping off a cliff." But in this case, you are probably at least as safe patching in hex as you are in using the Assembler.

Interesting side observations: Both the Intel assembler ASM86 and the Digital Research assembler RASM86 performed all of the operations in Listing Two correctly. They also take up about half as much space on the disk as the Microsoft Assembler and run much faster. RASM86 is now available for PC-DOS as part of the Digital Research "Programmer's Utilities" package.

Highly Recommended Software

Those of you who like RAM-disks (and who doesn't?) should consider the purchase of JETDRIVE for DOS 2.0 from Tall Tree Systems in Palo Alto, California. The program is solid as a rock, works with any RAM expansion board, costs only $40, and is even supplied with source code. Also included is the JET utility which transfers files between disks at a rate guaranteed to astonish the most jaded programmer. If you thought the PC-DOS COPY function was fast compared to CP/M's PIP, you won't believe JET.

And for those small software houses wrestling with the problem of how to supply all the eighty zillion 5¼-inch, soft-sectored disk formats, inexpensive relief is at hand. The program UNIFORM, from Micro Solutions in DeKalb, Illinois, can initialize disks and copy files to or from disks for some 40 different computers including such formerly difficult animals as the QX-10 and the Otrona. It will also transfer files between a CP/M disk and any of the MS-DOS single- or double-sided formats. UNIFORM currently runs on the various Kaypro models but is being ported to other machines. The best news of all is that UNIFORM costs only $50! At that price, it's worthwhile to buy a Kaypro just to use it – the alternatives are hideously expensive. DDJ

Reader Ballot
Vote for your favorite feature/article.
Circle Reader Service **No. 199.**

16-Bit Toolbox

Listing One

```
                                name    optest
                                page    55,132
                                title   'OPTEST - test Microsoft Assembler operators'
                        ;
                        ; Show operation of various operators and demonstrate some
                        ; notational idiosyncracies in the Microsoft IBM PC Assembler.
                        ;
                        ; Ray Duncan, November 1983
                        ;
                        ;
                        ; Every other assembler I have ever encountered will display
                        ; the true hex equivalent of an equate in the object code column.
                        ; The Microsoft Assembler, however, shows a signed hex integer!
                        ;
      =-0001            neg_one equ     -1      ;should display as FFFF


                        ; For unknown reasons, the Microsoft Assembler also fails to display
                        ; the byte swapped nature of some 16 bit values.

      0000  0001                dw      1


                        ; The Microsoft Assembler manual says nothing about the SHR
                        ; and SHL operators.  However, an example on page 4-21 states
                        ; that the operation 101b shl (2*2) should return 01010000b or 50H,
                        ; implying that the order of arguments is data SHL shift_count.

      0002  0080                dw      101b shl (2*2)

                        ; Since Microsoft's own example doesn't work with their assembler,
                        ; by experimenting we find that the expected order of arguments is
                        ; shift_count SHL data. This conflicts with the Intel specification.

      0004  0001                dw      0 shl 1
      0006  0002                dw      1 shl 1
      0008  0004                dw      2 shl 1

                        ; The SHR operator doesn't work correctly.  Apparently gives
                        ; the same results as SHL...

      000A  0004                dw      2 shl 1
      000C  0004                dw      2 shr 1

```

(Continued on next page)

(Listing continued, text begins on page 84)

Listing One

```
                                  ; Except when it gives no result at all...

      000E  0010                        dw      1 shl 8
      0010  0000                        dw      1 shr 8

                                  ; SHR may even give different results with equivalent data

      0012  FFFF                        dw      16 shr -1
      0014  0000                        dw      16 shr 0ffffh

                                  ; Sometimes the SHL operator seems to perform a "logical shift"

      0016  FFFE                        dw      1 shl -1
      0018  8000                        dw      15 shl -1

                                  ; other times, it appears to perform a circular shift

      001A  FFFF                        dw      16 shl -1

                                  ; The EQ operator doesn't work properly

      001C  FFFF                        dw      1 eq 1
      001E  FFFF                        dw      1 eq -1

                                  ; The EQ operator can give different results with equivalent data

      0020  FFFF                        dw      1 eq -1
      0022  0000                        dw      1 eq 0ffffh

                                  ; The NE operator is similarly afflicted

      0024  0000                        dw      1 ne 1
      0026  0000                        dw      1 ne -1
      0028  FFFF                        dw      1 ne 0ffffh

                                  ; The LE, LT, GE, and GT operators give confusing results

      002A  0000                        dw      -1 lt 1
      002C  FFFF                        dw      -1 le 1
      002E  0000                        dw      -1 gt 1
      0030  FFFF                        dw      -1 ge 1

                                  ; Again, these operators can give different results with
                                  ; equivalent data

      0032  FFFF                        dw      1 ge -1
      0034  0000                        dw      1 ge 0ffffh

                                  ; The NOT operator fails miserably on some signed integers

      0036  0002                        dw      not -1

```

```


                               ; Similarly, the OR operator flubs with signed integers

 0038  0001                         dw      -1 or 0
 003A  FFFF                         dw      0ffffh or 0

                               ; The XOR operator apparently works as an Inclusive OR
                               ; instead of Exclusive OR

 003C  0000                         dw      0 xor 0
 003E  0001                         dw      1 xor 0
 0040  0001                         dw      0 xor 1
 0042  0001                         dw      1 xor 1

                                    end
```

End Listing One

Listing Two

```
                                    name     closer
                                    page     55,132
                                    title    'CLOSER - show bug in PC-DOS function 10H'
```

(Continued on next page)

16-Bit Toolbox

(Listing continued, text begins on page 84)

Listing Two

```
                        ;
                        ; This program demonstrates a subtle but dangerous bug in the
                        ; PC-DOS Close File function 10H.  If a Close request is issued
                        ; using a file control block that has not been previously
                        ; activated by a successful Open command, the file's length
                        ; will be truncated to zero and the clusters previously assigned
                        ; to the file are left floating.
                        ;
                        ; Ray Duncan, November 1983

 = 000D                 cr      equ     0dh        ;ASCII carriage return
 = 000A                 lf      equ     0ah        ;ASCII line feed

0000                    cseg    segment para public 'CODE'

                                assume  cs:cseg,ds:data,es:data,ss:stack

0000                    closer  proc    far
0000  1E                        push    ds         ;save DS:0000 for final
0001  33 C0                     xor     ax,ax      ;return to PC-DOS
0003  50                        push    ax

0004  B8 ---- R                 mov     ax,data    ;make our data area
0007  8E D8                     mov     ds,ax      ;addressable
0009  8E C0                     mov     es,ax

                                                   ;now create file QUACK.DAT
000B  B4 16                     mov     ah,16h
000D  BA 013A R                 mov     dx,offset fcb
0010  CD 21                     int     21h
0012  0A C0                     or      al,al      ;create successful?
0014  75 4D                     jnz     closer8    ;no,jump
0016  B4 09                     mov     ah,9       ;yes,print success message
0018  BA 0000 R                 mov     dx,offset msg1
001B  CD 21                     int     21h

                                                   ;now set the record length
                                                   ;to 1024 bytes and write
                                                   ;random data into the
                                                   ;file (using default DTA)
001D  C7 06 0148 R 0400         mov     word ptr fcb+14,1024
0023  B4 15                     mov     ah,15h
0025  BA 013A R                 mov     dx,offset fcb
0028  CD 21                     int     21h
002A  0A C0                     or      al,al      ;was write successful?
002C  75 35                     jnz     closer8    ;no,jump
002E  B4 09                     mov     ah,9       ;yes,print success message
0030  BA 001B R                 mov     dx,offset msg2
0033  CD 21                     int     21h

                                                   ;now close the file so the
                                                   ;directory will be updated
```

```
0035  B4 10            mov     ah,10h
0037  BA 013A R        mov     dx,offset fcb
003A  CD 21            int     21h
003C  0A C0            or      al,al     ;close operation successful?
003E  75 23            jnz     closer8   ;no,jump
0040  B4 09            mov     ah,9      ;yes,print success message
0042  BA 0041 R        mov     dx,offset msg3
0045  CD 21            int     21h

                                         ;now clear out the file control
                                         ;block except for the filespec,
                                         ;as though the file had never
                                         ;been opened, and then request
                                         ;another close operation.

0047  BF 0146 R        mov     di,offset fcb+12
004A  B9 0019          mov     cx,25
004D  32 C0            xor     al,a'
004F  FC               cld
0050  F3/ AA           rep stosb         ;this zeros out the fcb

0052  B4 10            mov     ah,10h    ;now close file again
0054  BA 013A R        mov     dx,offset fcb
0057  CD 21            int     21h
0059  0A C0            or      al,al     ;check status
```

(Continued on next page)

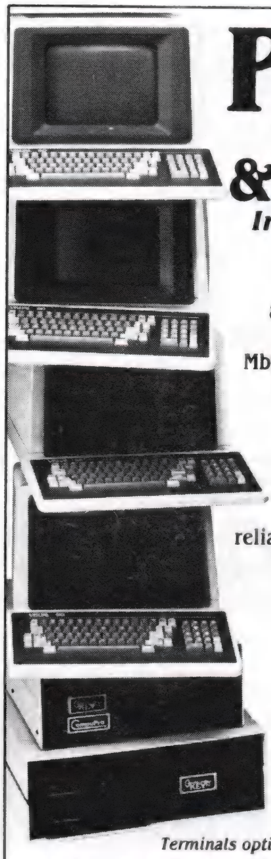
Listing Two

```
005B  75 06                     jnz     closer8   ;bad status,jump

                                                  ;status ok, print final
                                                  ;message to inspect directory
005D  BA 0064 R                 mov     dx,offset msg4
0060  EB 04 90                  jmp     closer9

0063                    closer8:                  ;come here if unexpected
                                                  ;failure of disk operation
0063  BA 0111 R                 mov     dx,offset msg5

0066                    closer9:                  ;print message and exit
0066  B4 09                     mov     ah,9
0068  CD 21                     int     21h

006A  CB                        ret               ;far return to PC-DOS

006B                    closer  endp

006B                    cseg    ends


0000                    stack   segment  para stack 'STACK'
0000      40 [                  db       64 dup (?)
                ??
                     ]

0040                    stack   ends


0000                    data    segment  para public 'DATA'

0000  0D 0A             msg1    db       cr,lf
0002  46 69 6C 65 20 51         db       'File QUACK.DAT created'
      55 41 43 4B 2E 44
      41 54 20 63 72 65
      61 74 65 64
0018  0D 0A 24                  db       cr,lf,'$'

001B  0D 0A             msg2    db       cr,lf
001D  31 30 32 34 20 62         db       '1024 bytes written into QUACK.DAT'
      79 74 65 73 20 77
      72 69 74 74 65 6E
      20 69 6E 74 6F 20
      51 55 41 43 4B 2E
      44 41 54
003E  0D 0A 24                  db       cr,lf,'$'

0041  0D 0A             msg3    db       cr,lf
0043  46 69 6C 65 20 51         db       'File QUACK.DAT closed properly'
      55 41 43 4B 2E 44
      41 54 20 63 6C 6F
```

```
         73 65 64 20 70 72
         6F 70 65 72 6C 79
0061     0D 0A 24                          db      cr,lf,'$'

0064     0D 0A 0A              msg4        db      cr,lf,lf
0067     53 65 63 6F 6E 64                 db      'Second close operation requested
         20 63 6C 6F 73 65
         20 6F 70 65 72 61
         74 69 6F 6E 20 72
         65 71 75 65 73 74
         65 64 20
0088     6F 6E 20 66 69 6C                 db      'on file QUACK.DAT'
         65 20 51 55 41 43
         4B 2E 44 41 54
0099     0D 0A                             db      cr,lf
009B     49 6E 73 70 65 63                 db      'Inspect length of file QUACK.DAT '
         74 20 6C 65 6E 67
         74 68 20 6F 66 20
         66 69 6C 65 20 51
         55 41 43 4B 2E 44
         41 54 20
00BC     6F 6E 20 64 69 72                 db      'on directory listing'
         65 63 74 6F 72 79
         20 6C 69 73 74 69
         6E 67
00D0     0D 0A                             db      cr,lf
```

(Continued on next page)

Circle **no. 61** on reader service card.



Circle **no. 37** on reader service card.

16-Bit Toolbox

(Listing continued, text begins on page 84)

Listing Two

```
00D2  74 68 65 6E 20 72              db      'then run CHKDSK with /F switch to
      75 6E 20 43 48 4B
      44 53 4B 20 77 69
      74 68 20 2F 46 20
      73 77 69 74 63 68
      20 74 6F 20
00F4  72 65 63 6F 76 65              db      'recover lost disk clusters'
      72 20 6C 6F 73 74
      20 64 69 73 6B 20
      63 6C 75 73 74 65
      72 73
010E  0D 0A 24                       db      cr,lf,'$'

0111  0D 0A                  msg5    db      cr,lf
0113  55 6E 65 78 70 65              db      'Unexpected failure of disk operation'
      63 74 65 64 20 66
      61 69 6C 75 72 65
      20 6F 66 20 64 69
      73 6B 20 6F 70 65
      72 61 74 69 6F 6E
0137  0D 0A 24                       db      cr,lf,'$'

                                                     ;file control block
013A  00                     fcb     db      0       ;use default drive
013B  51 55 41 43 4B 20              db      'QUACK   DAT'
      20 20 44 41 54
0146     19 [                        db      25 dup (0)
                  00
                       ]


015F                         data    ends

                                     end     closer
```

End Listing Two

MOTOROLA 68000 STRUCTURED MACRO CROSS ASSEMBLER

for CP/M-80* (8" SSSD)

EXORmacs† Compatible

$200

Manual
$20

Linker included • Source Available

farbware
1329 Gregory
Wilmette, Ill. 60091

*Trademark of Digital Research
†Trademark of Motorola Inc

Circle **no. 28** on reader service card.

MEMORY FREEDOM FOR $99.50
LOVLY-3

The Overlay Linker from **easy**

LOVLY-3 is a powerful development tool for the implementation of large Fortran programs that exceed the memory capacity of a microcomputer.

LOVLY-3 is a linker with program segmentation and overlay capabilities. It accepts object files generated by MicroSoft's F80 compiler and M80 assembler, and produces executable programs which run under Digital Research's CP/M operating systems.

- No modification of the source program being segmented; implicit loading of segments.
- No limit on total program size; segment size up to 64k.
- Multiple entry segments; up to 200 modules; 8 level deep overlay structure.
- Complete user documentation with example.

LOVLY-3 runs on Zilog Z80 and Intel 8080/ 8085 based micros with 56K memory and CP/M 2.x.

Disk formats include: 5" NorthStar Advantage and Apple II. Other disks available soon.

Terms: Check or COD.

easy
Engineering Application Systems
P.O. Box 10998 #529
Austin, TX 78766-1998

Circle **no. 25** on reader service card.

eXchanger CP/M ⟷ ISIS Emulator

Now move files and programs between your CP/M-80 system and your Intel Series I or II MDS! The ICX package provides complete bidirectional file conversion capability, and even **allows execution** of ISIS-II programs under CP/M using the ISE emulator. The ICX Package is composed of the following two programs:

ICX A Deluxe bidirectional file conversion utility which works with your CP/M system and an 8" floppy drive to provide complete manipulation of an ISIS-II diskette. Takes directories, deletes files, and even initializes a blank disk with the ISIS file structure. Complete C source included. **$89**

ISE An ISIS-II Emulator which allows ISIS programs to run on any CP/M-80 system. Support for all ISIS-II system and monitor calls makes your CP/M micro look like an MDS! Supports banked memory. Complete MAC source included **$89**

Complete ICX Package (ICX & ISE) $175

Supplied on Single Density 8" Disk
CP/M Digital Research Inc ISIS-II Intel Corp

Western Wares
Box C
Norwood CO 81423
(303) 327-4898

Circle **no. 84** on reader service card.

ICs PROMPT DELIVERY!!!

SAME DAY SHIPPING (USUALLY)

DYNAMIC RAM		
256K	150 ns	$85.00
64K	200 ns	5.97
64K	150 ns	6.09
64K	120 ns	6.97
16K	200 ns	1.56
EPROM		
27128	300 ns	$19.20
2764	250 ns	6.30
2732	450 ns	4.25
2716	450 ns	3.60
2532	450 ns	4.75
STATIC RAM		
6264P-15	150 ns	$35.97
6264LP-15	150 ns	40.00
6116P-3	150 ns	5.75

Prices 12/14/83

MasterCard VISA or UPS CASH COD
Factory New, Prime Parts µP∞
MICROPROCESSORS UNLIMITED
24,000 South Peoria Ave.
BEGGS, OK. 74421
(918) 267-4961

Prices subject to change. Please expect higher prices on some parts due to world wide shortages. Call for volume prices. Subject to available quantities. Shipping & Insurance extra. Cash discount prices shown. Federal Express Standard Air @ $5.99! Orders received by 6 PM CST can be delivered to you by the next morning. We try harder.

Circle **no. 45** on reader service card.

HEATH. . .OSBORNE. . .KAYPRO. . .ANY CP/M*-80

Tired of 16-bit Snobbery?

C/NIX-80™

Brings

UNIX* Power to CP/M*-80

Features	Prices	
Hierarchical Directories	C/NIX™-80	65$
UNIX*-like Shell	Manual Only	20$
I/O Redirection & Pipes		
On-Line Documentation	Specify Disk Format	
GREP, CAT, LS, CHDIR, etc.	Requires 48K CP/M-80	

C/Craft™, Attn: T. Taft
22 Downing Road, Lexington, MA 02173
(617) 862-8177

*UNIX is a trademark of Bell Laboratories
CP/M is a trademark of Digital Research

Circle **no. 18** on reader service card.

BRIDGE GRAPHICS

PLOTPAK is a complete plotting library that runs under FORTRAN-80 and performs a variety of functions: ***windowing, linear print arrays, automatic polygon drawing, annotations, plotting symbol/line selection, labeling, coordinate conversions.***

PLOTPAK can drive a screen and plotter simultaneously while maintaining a window which allows you to communicate back to your FORTRAN program. PLOTPAK includes your choice of the following drivers:

SCREENS

- MicroAngelo MA 512, 520
- ADM + Retrographics
- TEK 4010 Compatible Terminals
- Otrona ATTACHE Portable Computer

PLOTTERS

- H. P. Plotters 7225B & 7470
- Houston Instruments DMP-4

PLOTPAK (.REL file) + two drivers	**$190**
PLOTAK (source code) + two drivers	**$240**

BRIDGE
Computer Company
DIVISION OF SEA DATA CORPORATION
ONE BRIDGE ST., NEWTON, MA 02158
PHONE (617) 244-8190

Get the power of your Z80, and the elegance of direct access to CP/M functions from your high level programs with

SYNLIB *utility library*

SYNLIB *consists of MICROSOFT compatible object code that may be called from any high level language that uses MICROSOFT parameter passing conventions.*

SYNLIB *gives you extremely powerful array and buffer manipulation using the Z80 LDIR instruction; program access to the CP/M CCP console command line; high speed disk block I/O; a high quality random number generator; hex to ASCII conversion optimized by special Z80 instructions; program chaining; and more.*

And, because our programmer abhors a vacuum, each 8" floppy comes packed with some of the most valuable public domain software, including available source, absolutely free. You get ***SWEEP****, a menued disk utility that makes a computer phobe a systems programmer;* ***UNSPOOL****, so you can print* ***and*** *use your computer without buying an expensive buffer; /, to get multiple commands on a line;* ***MODEM7****, so that you too can join the free software movement; and many others.*

SYNLIB $50.00 8" SSSD CP/M format
SOURCE: $100.00
Licensing for commercial use available.
SYNTAX CONSTRUCTS, Inc.
14522 Hiram Clarke, Houston, Texas 77045
(713) 434-2098

CP/M is a registered trademark of Digital Research, Inc.
Microsoft is a registered trademark of Microsoft Corp.

Circle **no. 79** on reader service card.

JVS–FORTH $10

- SUPPORTS BYTE, INTEGER, AND REAL DATA
- COMPLETE ACCESS TO CP/M
- REQUIRES ASSEMBLER/LINKER BELOW

BASIC COMPILER $20

- SUPPORTS INTEGER REAL AND STRING DATA
- 25 STATEMENT TYPES
- GENERATES 8080 ASSEMBLER CODE
- RUN TIME LIBRARY SOURCE AVAILABLE ($50 EXTRA)
- MANUAL ONLY $5 (REF WITH SOFTWARE PURCHASE)
- REQUIRES ASSEMBLER/LINKER BELOW

ASSEMBLER/LINKER $10

- SYMBOLIC 8080 RELOCATABLE ASSEMBLER
- LINK EDITOR (LIBRARY SEARCH CAPABILITIES)
- LIBRARIAN FOR LINK EDITOR LIBRARIES

FREE BROCHURE AVAILABLE

REQUIRES CP/M-80 V2.2 AND 32K
8" 3740 SSSD OR APPLE 5¼" 16 SECTOR
DISK FORMATS ONLY

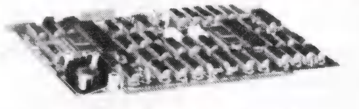
WHEN ORDERING ADD $10 FOR
SHIPPING HANDLING AND MEDIA COSTS
MASS SHIPMENTS ADD 5% SALES TAX
MAKE CHECK OR MONEY ORDER PAYABLE TO

JV SOFTWARE
P.O. BOX 684·NEWTON, MA 02162

CP/M is a trademark of Digital Research Inc
APPLE is a trademark of Apple Computer Inc

Circle **no. 36** on reader service card.

80 CHARACTER VIDEO BOARD
- **WORDSTAR/dBASE II OPTION**
- **TYPE AHEAD KEYBOARD BUFFER**



- 25 LINE NON-SCROLL OPTION
- Z80 CPU and 8275 CRTC S-100
- CHARACTER GRAPHICS
- ADAPTABLE SOFTWARE
- ORDER ASSEMBLED & TESTED OR PRE-SOLDERED (ADD YOUR IC's)

VDB – A2 bare board from $49.50

Simpliway PRODUCTS CO.
(312-359-7337)

P.O. BOX 601, Hoffman Estates, IL 60195
add $3.00 S&H, 3% for Visa or Mastercard
Illinois Res. Add 6% Sales Tax

WORDSTAR is a trademark of MicroPro INTERN'L CORP.
dBASE is a trademark of ASHTON-TATE CORP.

Circle **no. 70** on reader service card.

C/UNIX PROGRAMMER'S NOTEBOOK

by Anthony Skjellum

In the first installment of this column, I proposed a standard for the layout of C code (see *DDJ* No. 84, October 1983). Significant reader response was received concerning this subject, the vast majority of which was in favor of the concept of a C layout standard. In this column, I will present reader comments concerning the layout standard and discuss some modifications and additions to the proposed standard based on reader suggestions.

Some comments were also received concerning my discussion of runtime library and linkage format incompatibilities. Discussion of these points will be left for a future column.

Questions of White Space

Several readers took exception to a particular point in the proposed layout standard, 4F, which states, "No white space character is placed between a keyword (e.g., *if*) and its parenthesized argument." David D. Clark of State College, Pennsylvania, writes:

"In general, I like your coding standard suggestions. My only strong objection is to your idea to leave out spaces between reserved words [and their arguments]. It makes them look like function invocations."

Tim Smith of Evanston, Illinois, notes:

" . . . I think that a single space between a function name and the initial opening parenthesis, or after '*if*s' and '*else*s,' looks better . . . ."

Guy Scharf of Mountain View, California, writes:

"4f. I have a strong preference to *always* put a space between a reserved word (e.g., *if, while*) and its parenthesized argument. This adds legibility for me."

Finally, Charlie Brady of New South Wales, Australia, writes:

"The only real beef I have with you is the formatting of keywords and their parenthetical expressions. I can see no reason to depart from Kernighan and Ritchie on this point, and a number of reasons for maintaining their convention. Firstly, a flow control construct is semantically distinct from a function call, and a formatting difference is a reasonable way of distinguishing them. Secondly, the formatting difference simplifies the use of a text editor for such tasks as constructing structure charts. Third, your recommendation departs from at least three extant recommended standards, namely Kernighan and Ritchie, Thomas Plum (*C Standards and Guidelines*, Plum-Hall, 1981) and Tim Lang ('Formatting C,' *AUUGN*, Vol. 4, No. 1, Jan. 1982. Enclosed)." (I want to thank Mr. Brady for including a copy of the Lang article with his letter. The C standard proposed there is very compatible with the one I have proposed.)

After considering the above remarks, I have come to the conclusion that the space really does serve a useful purpose. Therefore, I suggest that point 4f should be changed to read: "A single white character is (optionally) placed between a keyword (e.g., *if*) and its parenthesized argument." (Making the white space character optional is another point for debate.) I think it should be optional but recommended. I don't think that adding a space for function call invocations would be beneficial, as suggested by Mr. Smith.

Another question concerning white space insertion comes in connection with argument lists. The original standard does

```
(a)  if( (a == 1) && (b == 2) && (c == 3) && ( (d == 4) || (d == 5) ) )
     {
              /* operations performed if conditional true */
              . . .
     }

(b)  if       /* we make multi-line expression look like a block */
     (
              (a == 1) &&
              (b == 2) &&
              (c == 3) &&
              ( (d == 4) || (d == 5) )
     )
     {
              /* operations performed if conditional true */
              . . .
     }
```

Figure 1.

(Under section 4)

b. Binary operators (e.g., +, -, /, but not —> and .) and assignment operators (e.g., =, *=, and &=) are delimited by white space.

g. Parentheses should be adjacent to the argument(s) which they enclose.

h. A comma is bound to the argument which precedes and should be followed by a single space.

i. Operators such as —> and . (used in pointer references) directly bind to their arguments with no intervening spaces.

Table 1.

not indicate if spaces should be included. It is my opinion that a comma should be directly adjacent to the argument that it follows, and that a single white space should follow each comma to add legibility. I am also convinced that parentheses should be adjacent to the argument(s) they enclose. Thus (in agreement with Tim Lang's article mentioned above), I would write:

```
        x = atan(sin(y));
```

and not

```
        x = atan( sin( y ) );
```

Yet another point not previously mentioned is that binary operators should be delimited by white space. Thus, the following statement lacks sufficient white space:

```
        v = sin(1n(1.0+x));
```

while this expression is properly formed:

```
        v = sin(1n(1.0 + x));
```

Finally, section 4 needs to be updated to include a style specification for pointer references. I think that operators "." and "—>" should not be delimited by spaces from the objects which they act on. This point and the three above are formalized in Table 1 (page 94) as additions to section 4.

The Lang article points out a circumstance under which point 4g need not be followed. This occurs when very complicated conditional expressions of the form "**keyword** (expr)" are split over several lines. In Figure 1 (page 94), for example, instead of using a crowded expression (Figure 1a), a more readable form is selected (Figure 1b). Note that in Figure 1b the parentheses are placed on lines by themselves, since they bracket a multiline expression, much like braces enclose the statements of a block.

Another point of minor objection was the tabulation method specified by the standard (point 1a). Steve Newberry of Los Altos, California, states:

> "Upon one point I do feel compelled to argue with you, and that is the tab convention: The *depth* of the tab stop on a given page is of far less significance to the readability of that page than is the *consistency* of the depth. I *really* don't want to use different size tabs on the same page."

Tim Smith writes:

> "I personally follow most all of his suggestions on how to actually format the code on the line and page, with only two exceptions. I *always* use 4-space tab stops . . . . "

I agree that having a single tab size is the preferable way to write C code. Standard tabs give more openness to the code, and make various parts of a program easier to pick out. My rationale for large horizontal tabbing is the same as for vertical tabbing. I want the program's significant portions to stand out. However, I propose adding point 1b to the standard: "4-space tabs may be used in lieu of standard tabs in cases where a subprogram includes highly nested segments." I would also include point 1c: "Only one of the two tabbing conventions should be employed in any given program module."

Some currently available screen editors provide a feature called horizontal scrolling. With horizontal scrolling, the user views a window of the file in both the vertical and horizontal directions. Thus, files with lines longer than the display device may be handled intelligently. Under such circumstances, there is no real disadvantage to using standard tabs to any desired nesting depth, which is permitted under point 1a of the proposed standard.

Other Corrections

Charlie Brady noted an unnecessary point in section 3e which stated: "When a null block is used (e.g., '{ }'), it may appear on the same line as other statements (e.g., do { } while(expr);)." He writes:

> "Another minor point of disagreement concerns the use of the null block ({ }). This is never necessary, and I believe that the null statement (;) is clearer. It should be emphasized that the null statement *deserves* a line of its own. Your example:
>
> ```
>         do { } while(expr);
> ```
>
> is more simply written
>
> ```
>             while (expr)
>                     ;                "
> ```

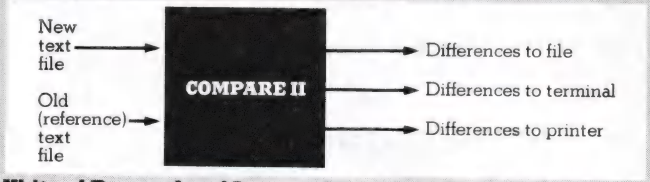
In accordance with Mr. Brady's remark, I propose replacing point 3e with the following: "The null block ({ }) can always be avoided. Instead of a null block, use the null statement." I also would add point 3f ii: "The null statement is always on a line by itself."

Documentation Standards

In addition to a code layout standard, James Halstead of Joliet, Illinois, has proposed a basic documentation standard. He writes:

> ". . . I strongly suggest that the original standards one through nine be renumbered two through ten so that the first and foremost standard may be inserted."

The documentation standard suggested by Mr. Halstead is presented in Table 2 (page 96) and I think it should be included in my proposed standard. However, I have decided to number the



0. Identification Description (I.D.) information must appear at the beginning of each C language source file.

   a. The recommended format is:

      i. Begin comment (/*).
      ii. Space.
      iii. Title (identification name normally = filename).
      *iv. Sub-title (i.e., what program system does this file belong to).
      v. Space.
      vi. Classification (see below).
      vii. Year.
      viii. Owner.
      ix. Status (see below).
      *x. Current Version number and brief history.
      xi. Date.
      xii. Functional/Structural Description in brief.
      *xiii. Portability synopsis.
      *xiv. Space.
      xv. End Comment (*/).
      *xvi. Space.

   b. The program classification (vi.) is one of the following:

      i. Public-domain.
      ii. Copyright.
      *iii. Copyright: released for non-commercial purposes.
      iv. Unclassified.
      v. Secret.
      vi. No classification.

   c. The program status (ix.) is one of the following:

      i. Outline.
      ii. Draft.
      iii. Test (alpha, beta, etc.).
      iv. Release.

Table 2.

10. Each function should contain the following minimum documentation:
   a. A general explanation of the function performed.
   b. Its name, and a description of its arguments including their types, and legal values.
   c. A description of the functional return value, if any.
   d. A list of non-standard functions used by the function.
   e. A list of external variables used and/or modified by the function.
   f. A description of the error handling characteristics of the function.
   g. A valid calling sequence example, if practical.

Table 3.

documentation standard as Section 0 in order to preserve the numbering of the current sections. (The slight additions I have made include an asterisk to indicate the addition.)

In addition to Section 0, I think a basic documentation standard is in order for functions as well. Such a standard is presented in Table 3 (page 96). I have placed this under Section 10, since I think of function documentation as a separate task from module documentation as described in Section 0.

Other Proposals

Several other readers made suggestions for the standard. I think that Tim Smith proposes several which merit discussion. They are presented here with the point numbers they receive as part of the standard:

5c. Don't nest comments, even if your preprocessor/compiler allows it.

6f. If there are many declarations, whether one line or many, alphabetize them.

9g. Restrict variable and function names to seven well-chosen characters, even if your compiler allows more.

Steve Newberry writes the following about standards:

"I applaud your interest in establishing a standard format convention for C programs. However I feel that your effort would have more impact if tied to support of Tom Plum's book, *C Programming Standards and Guidelines, Version U (Unix and offspring)*, Edition 3: Jan. '82. Presented in this manner, your proposed formatting standard would be seen as a consistent extension of a more general set of standards already in wide circulation."

Other Points of View

Although most readers were favorable to the idea of a C format standard, Douglas M. Potter of Seattle, Washington, writes:

"I'm afraid I don't see much advantage of your proposed standard over theirs [Kernighan and Ritchie]. In both cases, the size of the indent is too large. I always run out of room on the right side with a tab-sized indent. I also find that nobody uses enough white space."

John F. Draffen of Texas City, Texas, wrote me a detailed letter on why he didn't like the idea. He writes:

"I am writing to express my objections . . . In the first place, I do not think a standard of this type is either necessary or desireable. The layout has nothing to do with portability which to my mind is the only excuse for a standard. It seems to me that it is hard enough to get people to agree on necessary standards.

"In the second place, I do not agree with many of your suggestions on style.

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 756 MENLO PARK, CA

POSTAGE WILL BE PAID BY ADDRESSEE

Dr. Dobb's Journal
For Users of Small Computer Systems

P.O. BOX E
MENLO PARK, CA 94026

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 756 MENLO PARK, CA

POSTAGE WILL BE PAID BY ADDRESSEE

Dr. Dobb's Journal
For Users of Small Computer Systems

P.O. BOX E
MENLO PARK, CA 94026

Dr. Dobb's Journal

For Users of Small Computer Systems

Reader Service Card

To obtain information about products or services mentioned in this issue, circle the appropriate number listed below. Use bottom row to vote for best article in issue. This card is valid for 90 days from issue date. Use only one card per person.

February 1984, No. 88

Name ______________________ Address ______________________

Phone ( ) ______________________ City/State ______________________ Zip __________

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27
28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54
55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81
82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108
109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135
136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162
163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187 188 189
Articles: 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214

Comments:

Purchase of Magazine:

1 □ Subscription
2 □ Computer Store
3 □ Newsstand
4 □ Bookstore
5 □ Passed on by friend/colleague
6 □ Other __________

Dr. Dobb's Journal
For Users of Small Computer Systems

Reader Service Card

To obtain information about products or services mentioned in this issue, circle the appropriate number listed below. Use bottom row to vote for best article in issue. This card is valid for 90 days from issue date. Use only one card per person.

February 1984, No. 88

Name ______________________ Address ______________________

Phone ( ) ______________________ City/State ______________________ Zip __________

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27
28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54
55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81
82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108
109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135
136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162
163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187 188 189
Articles: 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214

Comments:

Purchase of Magazine:

1 □ Subscription
2 □ Computer Store
3 □ Newsstand
4 □ Bookstore
5 □ Passed on by friend/colleague
6 □ Other __________

PLACE
STAMP
HERE
The Post Office
will not deliver
mail without postage

Dr. Dobb's Journal
For Users of Small Computer Systems

ADVERTISING DEPARTMENT
P.O. BOX E
MENLO PARK, CA 94026

PLACE
STAMP
HERE
The Post Office
will not deliver
mail without postage

Dr. Dobb's Journal
For Users of Small Computer Systems

ADVERTISING DEPARTMENT
P.O. BOX E
MENLO PARK, CA 94026

Dr. Dobb's Journal
NOW AT BIGGER SAVINGS!

~~$35.40~~
$25.00

If you take advantage of this special offer you save over $10 off newsstand prices, that's a 30% savings!

______Please charge my: ____Visa ____MasterCard ____American Express
______Payment enclosed ______Bill me later

Card #______________________________ Exp. date__________

Signature ______________________________________

Name____

Address ____

City______________________ State______ Zip__________ Z2

Offer good in USA only. Foreign rates upon request. Please allow up to six weeks for first issue.
This offer good until March 31, 1984.

A publication of People's Computer Co., P.O. Box E, Menlo Park, CA 94026.

February 1984, No. 88

Dear Reader,

Dr. Dobb's has a long tradition of listening to its readers. We like to hear when something really helps or, for that matter, bothers you. In this hectic world of ours, however, it is often difficult to take the time to write a letter. This card provides you with a quick and easy way to correspond. Simply fill it out and drop it in the mail. We take care of the rest. Thanks for taking a few minutes to talk with us.

—Ed.

Which articles or departments did you enjoy the most this month? Why? (Please indicate order of preference.)

____

____

Comments or suggestions: _______________________

____

____

Name____

Address ____

____

One of the nice things about C that it shares with Fortran is its relative conciseness. I do not like to see code strung out unnecessarily. C does not interject unnecessary constructions, and I think that introducing unnecessary white space, excessive indentation, and meaningless comments is a kind of gingerbread that we can do without." Mr. Draffen's style of philosophy is listed in Table 4 (page 97).

I don't really agree with Mr. Draffen on several counts. First and foremost is that portability is not the sole subject of importance in programming. The ability to maintain, understand, correct, and enhance code is of great importance. To understand someone else's code (or your own code at a later date) requires some degree of formatting. Comments which seem less than essential to the programmer must sometimes be included for the sake of others. This is immensely important. It is often difficult for programmers to know how to comment their code, since they usually cannot know the level of sophistication of later readers. Thus, it is often better to include a few extra comments, than to comment code sparsely.

I suggested in my previous column that users should maintain their own code in the form that they prefer. However, code distributed to others could (and should) meet some minimum standard of neatness (i.e., formatting) and presentation. Some of this can be provided by a beautifier but most must be done by the programmer.

As one final note on C layout, I'm including an insightful paragraph which Tim Smith included in his letter. It suggests why so much C code is so poorly formatted and commented:

"I don't think, however, that you will ever get Unix wizards to follow these recommendations. I should have noted that I use Skjellum-like conventions when I'm writing micro-based applications. When I'm maintaining Unix sources I stick with the standard Unix conventions, which are pretty much Kernighan and Ritchie standard. Unix whizzes think that aligning curly braces is irrelevant, since "vi," the editor which 90% of them use, will always let you find the top or bottom match for any brace automatically. Also, and probably more important, Unix system guys always debug by staring at their CRTs, never from printouts (that's for COBOL programmers), and the goal is to reduce the number of lines of a function, so that as much of it as possible will fit on a screen. Seeing a *whole line* taken up by just an opening brace must drive them whacko, and some of them will even close blocks on the end of a line of code (yecch!)."

In this column, I have presented additions and corrections to the proposed layout standard, based on reader response. I have also presented the opposing point of view. Most of the letters received were positive, so it was difficult to include more dissenting remarks. I want to thank those who sent in their comments about the standard.

Reader Ballot
Vote for your favorite feature/article.
Circle Reader Service **No. 200.**

1. Punctuation should be used sparingly. The insertion of unnecessary white space should be avoided.
2. Block structures should be used, indicated by indentation. Excessive indentation should be avoided.
3. Comments and program code should be separated. Comments on the same line as code should be displaced far enough to the right that they do not obscure the code.
4. Comments should be meaningful. Comments that do no more than repeat what has already been said by the code should be avoided.

Table 4.

OF INTEREST

by Michael Wiesenberg

Networking PCs

PCterminal, from Santa Clara Systems, is an IBM-compatible personal computer with built-in local area network. Use it as an intelligent terminal in an IBM PC network, PCnet, instead of IBM PCs, to increase user stations at one-third the cost. You get monitor, keyboard, 8088, serial and parallel interface, four expansion slots, 64K RAM (expandable to 256), built-in networking, and connection for optional 5.25-inch floppy disk drive, for $1295. It runs either PC-DOS or Santa Clara Systems' version of MS-DOS, SCS-DOS. You can connect up to 16 PCterminals to one IBM PC or XT in a Santa Clara Systems network. PCnet, built into the terminals, lets PCs link with thousands of others through special adapter boards and software. "Remote execution" permits one terminal to run a command on another. **Reader Service No. 101.**

8086 C and Pascal

Whitesmiths has Release 2.2 of its **C** and **Pascal** compilers, adding 8086 architecture and Unix-style libraries. Whitesmiths now supports over 30 operating systems on five architectures: PDP-11 family, 8080 and Z80, MC68000, VAX family, and 8086. The last includes 8088, with optional 8087 math coprocessor support plus 80186 and 80286 instruction sets. On IBM PC, programs run under CP/M-86, DOS 1.x, and DOS 2.0. C native compiler is $550, Pascal native $1100, C cross compiler $1100, Pascal cross $1400. **Reader Service No. 103.**

The Digital Doctor

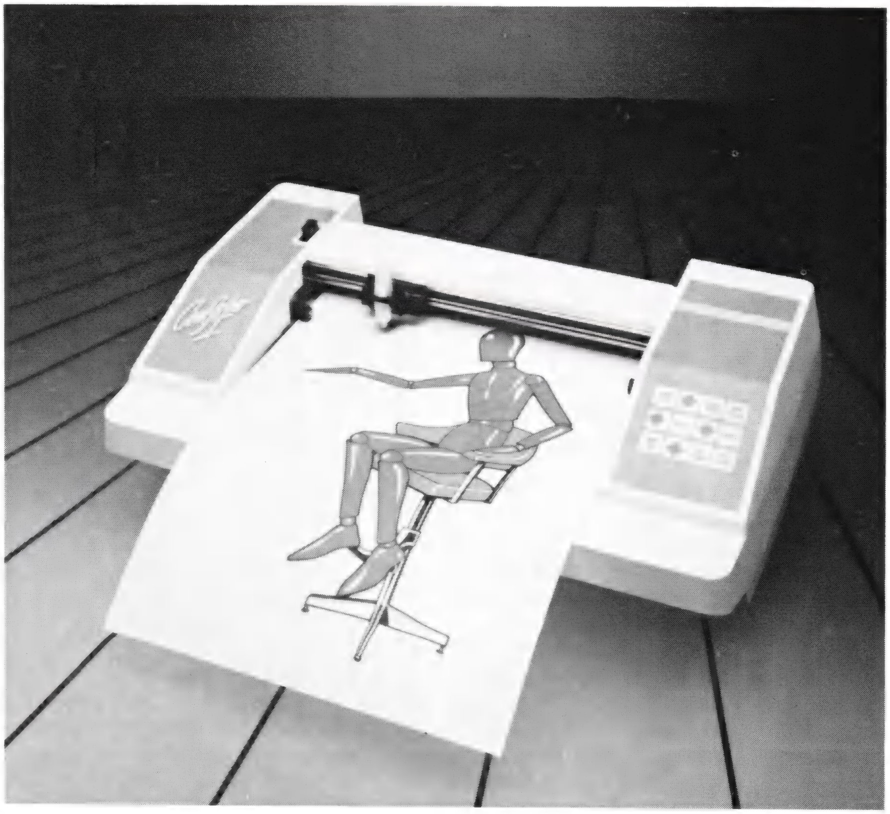
Not to be confused with the resident intern, the digital doctor makes house calls. Data Encore, a subsidiary of Verbatim, offers the **Datalife Disk Drive Analyzer**, a diagnostic software tool to check disk drive performance, testing head alignment, disk clamping, write/read accuracy, and disk speed. $39.95 for IBM PC, XT, and Apple. The latter includes enhanced graphics. When you return the warranty card, Data Encore will send you a free Datalife Twin Pack of minidisks or head cleaning kit. **Reader Service No. 105.**

Box It

You can make your IBM PC portable, or at least somewhat transportable, by putting it in a **Firebilt** case, for $181, and put the monitor and a printer in another case for $191. These are foam-padded luggage style cases made of high-impact plastic. Other cases made of plastic, aluminum, or fiberglass are available for scores of other computers, monitors, and printers, ranging in price from $65 to $315. **Reader Service No. 107.**

Computer Becomes Typewriter

Last time I described a product that turns electronic typewriters into printers. This time, to be fair, I mention a product that lets your printer become a typewriter, controlled by the keyboard of your computer. Hall Associates think that you do not want to be so inefficient as to call up your word processor just to type out one mailing label or a short note, and so they offer **TYPWTR** that lets your



Inexpensive Plotter

The **Model CR-1810 Comscriber I**, from Comrex International, is a lightweight, single-pen plotter that takes paper 8.5 inches wide and up to 10 feet long, and pauses to change pens. The price is $695. **Reader Service No. 119.**

MEGA-BYTE

SPECIALIZING IN HARD DISKS AND HIGH PERFORMANCE STREAMING TAPE UNITS

MD - 10 11 MEGABYTE (FORMATTED) 5 1/4" HARD DISK Including POWER! $1995

POWER!

Check out InfoWorld's Rave review of POWER Nov. 8/82)

55 CP/M UTILITIES IN ONE PACKAGE

...The super program that puts you in control of CP/M (or MS-DOS) and your winchester.

POWER automatically numbers disk files. Just pick file number to Copy, Erase, Reclaim, Type, etc.
...Your computer feeds the file names automatically. You do NO typing! NO typing error ever!
YOU DON'T NEED SYSTEM DISK IN ANY DRIVE.
No more BDOS ERROR!

YOU Test and Fix bad disks! Reclaim accidently erased files or programs! Single step thru memory up or down! Search, view, change memory or disk in a snap! See Status and File Size instantly! Verify check-sums for programs! Load or Save programs at any address.

55 prompted user-friendly funstions for housekeeping and a 120 page easy-read user's guide make POWER your most often used software. You'll use it every day!

Now, POWER! also includes a special program that lets you lock sensitive files, so that only you can access then. Without the secret PASSWORD which you can create and change at will, no prying eyes will ever know your secret file even exists.

SOME MAJOR POWER COMMANDS:

REN
TYPEA
SIZE
XUSER
SETRO
GROUP
SAVE
READGR
DUMPH

CM
USR 2
ERA
TYPEX
CHECK
TEST
SETWR
LOG
LOAD

WRITEGR
DUMPA
SEARCH
?
COPY
TYPEH
EXIT
USER
SETSYS

DISK
SPEED
WRITE
DUMPX
FILL
EX
DIR
TYPE
RUN

STAT
SETDIR
RECLAIM
DS
READ
DUMP
MOVE
JP

For this low price your winchester will be delivered completely assembled and tested, with drive controller, case, power supply, cabling, Z-80 interface and the best disk controller software on the market.

- The unique and simple interfacing system does not tie up existing ports.
- Significantly faster than other winchester subsystems which interface through the IEEE-488 port.
- CP/M drivers require minimum memory overhead (about 2K-other systems require as much as 6k)
- The MD-10 can read or write a 64K file in less than four (4) seconds.
- A network system will be available shortly which can support up to sixteen mixed types of computers from one MD-10 or larger disk subsystem.
- With POWER! Software files can be code word protected.
- Other systems available: 22 and 44 megabytes

MD-20 with 22MB formatted: $2895

MD-44 with 44MB formatted: $3995

- Up to eight (8) winchester subsystems can be interfaced to one computer.
- Software supports 32 different user areas per MD-10.
- Backing up hard disk files is simple with the special software which is provided with all subsystems.
- With a single hard disk installation, the MD-10 subsystems becomes units A and B with the standard drives being designed E and F if a second MD-10 is installed later it becomes units C and D.
- MD-10 or larger systems will interface with IBM PC or any Z-80 computer (CCS, APPLE (CP/M), ZENITH/HEATH, NORTHSTAR, GODBOUT, XEROX 820, Z-80/S100, ALSPA, or TRS-80 MOD II) using CP/M, OASIS, PCDOS 2.0, and Godbout Software supports both CP/M and MP/M816.
- Double density modifications are available allowing you to later increase the capability of an MD-10 to about 20 megabytes, MD-20 to 40, etc.
- **Full One-Year Warranty**

HIGH PERFORMANCE STREAMING TAPE

Unique! Simple! Compatible!

- **Field-proven Archive Sidewinder tape unit.**
- **Interfaces with ANY SASI/SCSI compatible hard disk peripheral.**
- **Full CRC error-checking**
- **Superb reliablity.**
- **Full One-Year Warranty.**
- **Fast (up to 20MB in less than 5 minutes, up to 40MB in less than 10 minutes).**

$1795.00 20MB

$1995.00 45/60MB

**$29.95 3M DC300XL
Streaming Tape Cartridges**

MEGA-BYTE

ORDER

Name	Qty.	Price	Total

VISA □ M/C □ COD □ Total Order ______

Card #__________ Exp. Date______ If Utah Tax ______

Signature__________ Total Enclosed ______

☎ TO ORDER OR FOR MORE INFORMATION:
(801) 257-7033 or mail to MEGA-BYTE INC. RT. 3 Box 35, Tremonton, Utah 84337
TERMS: Cashiers Check, VISA, M/C or COD.
Shipping charges added to all orders.

MasterCard VISA

NAME__________
COMPANY__________ TITLE__________
ADDRESS__________ CITY__________
STATE__________ ZIP__________
TEL (   ) __________

CP/M is a registered trademark of Digital Research. OASIS is a proprietary product of Phase One Systems Inc. Z-80 is a trademark of Zilog. MD-10 is a registered trademark of Media Distributing.

Circle no. 42 on reader service card.

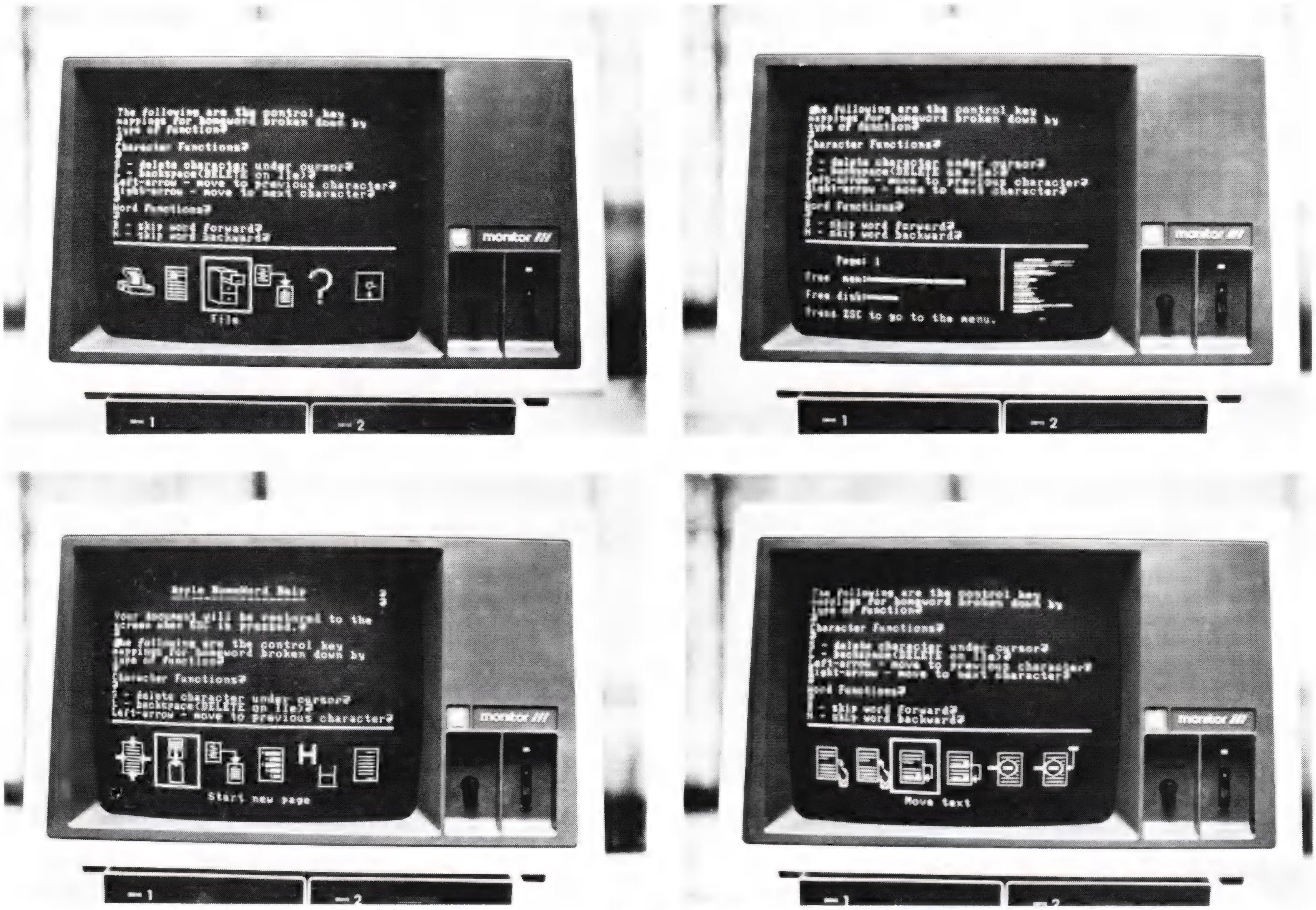
printer immediately track every keystroke. (What's wrong with using CP/M's Control-P?) You can have lines of 250 characters and word wrap; you can set margins and tabs. You need CP/M 2.x, 48K, and a printer that backspaces and does carriage returns without line feed. $29.95 plus $1.50 p. and h. **Reader Service No. 123.**

2068 and Adam Too

Looks like Timex has another winner with its 2068 Personal Color Computer. The under-$200 computer seems to address itself to all the difficulties users of the 1000 and 1500 had. The new machine has a "real" keyboard, not just sound but synthesizer capabilities, instant-load cartridges, high-resolution color graphics, joysticks, etc. What it lacks is a lot of software, although I understand Timex is adapting Sinclair Spectrum software as fast as they can. In the meantime, Softsync fills the gap with several games, **Cosmic Gorilla** ("masses of mutant mohair relentlessly stealing everything they can get their hands on"), **Gulpman** ("The cursed wormoids are out to get control of Gulpland, chasing its inhabitants out of their apple orchards. Eat as many apples as you can to get bonus points and use your lasers to stun the wormoids"), **Cyberzone**, a voice-actuated game, all on cassette and all $19.95. **Voice Chess** talks to you during play, recommending moves, if you wish, letting you switch sides, and analyzing moves, $24.95. Softsync also has **Zeus Monitor/Disassembler** and **Zeus Disassembler**, $19.95 each, indispensible for machine language programmers.

And if you need software for your Coleco Adam, Softsync has **Personal Accountant**, a double-posting bookkeeping program for household and small business use, $34.94 for cassette; **Model Diet**, computerized meal planning based on nutritional requirements, $29.95; **Dancing Feats**, a one-man joystick band that lets even beginners play synthesized music instantly accompanied by synchronized color display, $29.95.

Softsync also offers many of these programs for Atari, Commodore 64, and IBM PC. **Reader Service No. 121.**



Inexpensive Word Processor

For your inexpensive printers, you need inexpensive word processing software. **Homeword**, from Sierra On-Line, comes with software, book, and tutorial audio cassette. Features include icons, or pictures of operations such as a file cabinet for filing, a page of print for editing, a printer for printing, and an unorganized page with an arrow pointing to an organized one for formatting. The specifications for this product seem similar to other word processors, but less usual features include a HELP key and the division of the screen into three sections. The upper section is largest and holds the working text. The lower right section shows a replica of the page as it will appear printed. The lower left lists available memory and disk space. The price is $49.95 for Apple II, II+, and IIe (soon available for Commodore 64 and Atari). Also included is a 30-day money-back guarantee. **Reader Service No. 117.**

Computers Talk to Each Other and the World

The **MITE/86** data communications program, from Mycroft Labs, turns computers into intelligent terminals to use with The Source, Compuserve, and the like, and to transfer files between 8- and 16-bit computers. You get full control over all communications parameters (parity, baud rate, etc.), automatic logon, three binary file protocols (XMODEM, CLINK/CROSSTALK, and Hayes), and everything else you'd expect of a good communications package for CP/M, for $195. The system is preconfigured for Rainbow, PC, Victor, and other CP/M-86 systems and is also available for various CP/M1-80 systems. **Reader Service No. 111.**

Low-Cost Printers

Alphacom offers a 40-column dot matrix printer, the **Alphacom 42**, that interfaces with most computers, for $99.95, complete with interface cable, or $79.95 without cable. It has upper and lower case and recognizes standard ASCII control codes. Alphacom also offers an 80-column, 100 cps printer, the **Alphacom 81**, for $214.95, including cable. **Reader Service No. 115.**

Basic BASIC

A good language needn't cost hundreds of dollars. The **Nevada BASIC** interpreter for CP/M systems, by Ellis Computing, is $39.95 for the diskette and documentation. The language includes IF . . . THEN . . . ELSE, BCD math (no rounding errors), 8- or 12-digit precision, single and multiline user functions, full matrix operations, and a full-screen text editor which can be accessed by pressing the RETURN key when a runtime error occurs. **Reader Service No. 109.**

Quiet, Fast Printer

Blue Chip Electronics has a 40 cps, 132-column, daisy wheel printer, called **Model BCD-4015**, with an optically controlled print head that automatically inserts paper and adjusts the left margin. The noise level is 57 decibels, which, they claim, makes it one of the quietest daisywheel printers available. It's bidirectional, logic seeking, and takes over 100 different printwheels. In addition to proportional printing, boldfacing, and underlining, a tractor feed is standard. There is an optional cut sheet feeder. Centronics 8-bit parallel interface is standard, and serial RS-232C, IBM PC, and IEEE-488 are optional. Warranty and after sale service are provided by General Electric Instrument and Communications Service Department, so maintenance and repair are available at 26 GE service locations in the US and Canada. The suggested retail price is $1,895.00. **Reader Service No. 133.**

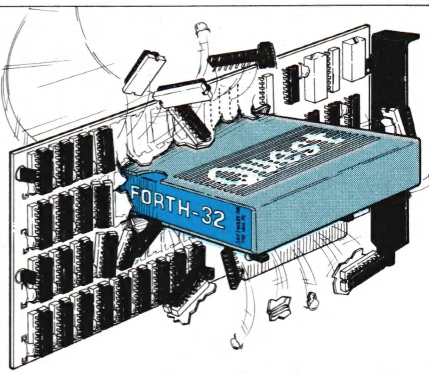
Fast Z80 Forth

q4th for CP/M, from Quanta Corporation, is claimed to be the "fastest Forth for Z80 in the known universe." It's a superset of Forth-79, is an interpreter and also compiles to Z80 machine code, and has a runtime debug trace with stack display $95 includes one year update and newsletter subscription. **Reader Service No. 125.**

DDJ

Reader Ballot
Vote for your favorite feature/article.
Circle Reader Service **No. 201.**

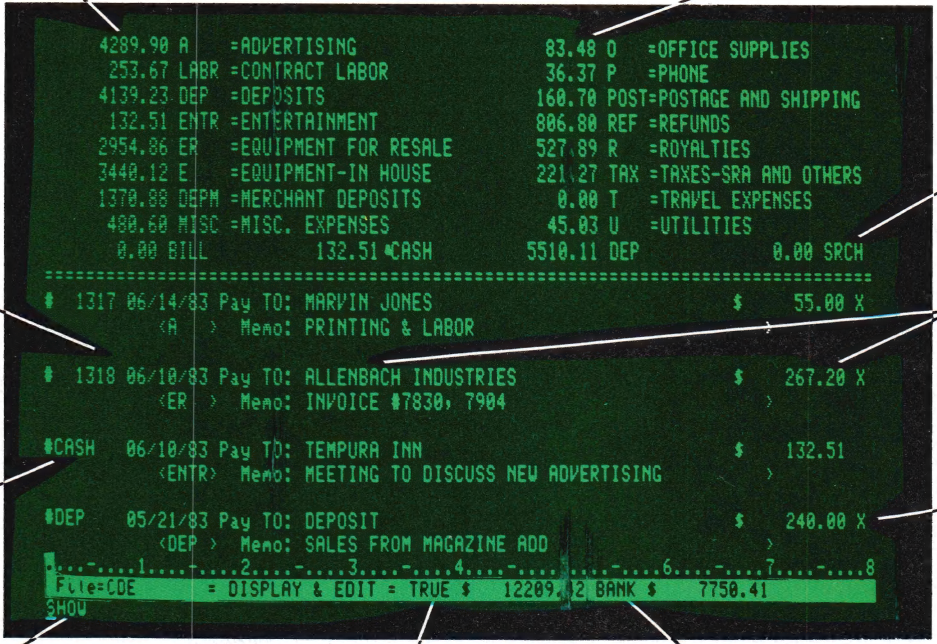
ADVERTISER INDEX



Q: What's easier than spending money?

A: Keeping track of it with... Checks & Balances

At last! A *full screen* editing checking and finance program so informative and easy to use you'll wonder how you lived without it! *Especially* at $\$49^{95}$



```
   4289.90 A    =ADVERTISING                     83.48 O   =OFFICE SUPPLIES
    253.67 LABR =CONTRACT LABOR                  36.37 P   =PHONE
   4139.23 DEP  =DEPOSITS                       160.70 POST=POSTAGE AND SHIPPING
    132.51 ENTR =ENTERTAINMENT                  806.80 REF =REFUNDS
   2954.86 ER   =EQUIPMENT FOR RESALE           527.89 R   =ROYALTIES
   3440.12 E    =EQUIPMENT-IN HOUSE             221.27 TAX =TAXES-SRA AND OTHERS
   1370.88 DEPM =MERCHANT DEPOSITS                0.00 T   =TRAVEL EXPENSES
    480.60 MISC =MISC. EXPENSES                  45.03 U   =UTILITIES
      0.00 BILL          132.51 CASH           5510.11 DEP               0.00 SRCH
==============================================================================
#  1317 06/14/83 Pay TO: MARVIN JONES                              $     55.00 X
          <A    >  Memo: PRINTING & LABOR                                 >

#  1318 06/10/83 Pay TO: ALLENBACH INDUSTRIES                      $    267.20 X
          <ER   >  Memo: INVOICE #7830, 7904                              >

#CASH   06/10/83 Pay TO: TEMPURA INN                               $    132.51
          <ENTR>   Memo: MEETING TO DISCUSS NEW ADVERTISING               >

#DEP    05/21/83 Pay TO: DEPOSIT                                   $    240.00 X
          <DEP  >  Memo: SALES FROM MAGAZINE ADD                          >
....-....1....-....2....-....3....-....4....-..........-....6....-....7....-....8
File=CDE        = DISPLAY & EDIT = TRUE $    12209.[illegible]2 BANK $     7750.41
SHOW
```

Sixteeen user-definable categories.

Category totals instantly updated with each new entry or change.

Screen always shows four entries. Use your cursor to add or change any entry anytime. Scroll forward and back through your register or instantly go to any entry.

Includes categories for bills, cash expenditures, deposits and service charges.

Checks and Balances aligns decimals and left justifies text.

For complete records, cash expenditures are included in category totals but not bank balance.

Entries which have cleared through your bank.

All commands in plain English.

True checking balance instantly updates with each entry.

Bank statement balance reflects only items checked off as having cleared your bank.

So easy to use! See your financial totals start to add up before your eyes in less than five minutes! Checks and Balances' cursor can move anywhere, anytime, to add or change entries, and all totals will instantly be updated on your screen when you change an amount or move a check to a different category. Checks and Balances gives you complete printed reports of the entire register, each category, or reports based on any entry in "Pay to" or "Memo"—even bills and uncashed or missing checks. File size is limited only by your disk capacity. Over 32,000 entries are possible.

Technical: 80 × 24 CRT with addressable cursor, reverse video optional. Z-80 processor, one or more disk drives with over 180k capacity. 56k RAM.

VISA

CDE SOFTWARE

2463 McCready Avenue
Los Angeles, CA 90039

MasterCard

Circle **no. 9** on reader service card.

(213) 661-2031

ENTER . . . THE FUTURE!

Introductory offer

$49.95

THIS IS THE PASCAL COMPILER EVERYBODY'S BEEN WAITING FOR... EVERYBODY EXCEPT THE COMPETITION!

	Turbo Pascal	IBM Pascal	Pascal MT+
PRICE	49.95	300.00	595.00
Compile & Link speed	1 second!!!	97 seconds	90 seconds
Execution speed	2.2 seconds	9 seconds	3 seconds
Disk Space 16 bit 8 bit	33K w editor! 28K w editor!	300K + editor Not Available	225K + editor 168K + editor
8 and 16 bit	YES	NO	YES
built-in editor	YES	NO	NO
Generate object code	YES	YES	YES
One pass native code compiler	YES	NO	NO
Locates RunTime errors directly in source code	YES	NO	NO

Benchmark data based on EightQueens in "Algorithms + Data Structures = Programs" by N. Wirth, run on an IBM PC.
Turbo Pascal is a trademark of Borland International. MT+ is a trademark of MT MicroSystems. IBM is a trademark of International Business Machines.

LEARN TO WRITE A SPREADSHEET

Our Introductory offer includes MICROCALC, a spreadsheet written in **Turbo Pascal**. It will be on your disk, and ready to run. And we've included the source code to show you exactly how a spreadsheet is written! (Introductory offer expires March 1, 1984)

Turbo Pascal includes a 250 page bound manual with extensive explanations and many illustrative examples.

Extended Pascal for your IBM PC, APPLE CP/M, MS DOS, CP/M 86, CCP/M 86 or CP/M 80 computer features:

- Full screen interactive editor providing a complete menu driven program development environment.
- 11 significant digits in floating point arithmetic.
- Built-in transcendental functions.
- Dynamic strings with full set of string handling features.
- Program chaining with common variables.
- Random access data files.
- Full support of operating system facilities.
- And much more.

ORDER YOUR COPY OF **TURBO PASCAL** TODAY TO TAKE ADVANTAGE OF OUR INTRODUCTORY SPECIAL.

For Visa and MasterCard orders call toll free

1-800-227-2400 X 968

IN CA: 1-800-772-2666 X 968

(lines open 24 hrs. a day, 7 days a week)

Dealer & Distributor Inquiries welcome.

Turbo Pascal $49.95 + $5.00 shipping per copy.

Check______Money Order______
VISA______Master Card______
Card #:____________________
Exp date:__________Shipped UPS

BORLAND
INTERNATIONAL

Borland International
4807 Scotts Valley Drive
Scotts Valley, California 95066

My system is: 8 bit______16 bit______
Operating system: CP/M 80______
CP/M 86____MS DOS____PC DOS____
Computer:______Disk Format:______
Please be sure model number and format are correct.
NAME:____________________
ADDRESS:____________________
CITY/STATE/ZIP:________________
TELEPHONE:__________________

California residents add 6½% sales tax. Outside North America add $15.00. Checks must be on a U.S. bank, and in U.S. dollars. Sorry, no C.O.D.